16ビット マイクロコンピュータ MC68000の使い方

小島 進 著 ●オーム社

写真提供：日本モトローラ(株)

図解 16ビット マイクロコンピュータ

MC 68000の使い方

小島 進 著

オーム社

は し が き

マイクロプロセッサが出現し，1ビット，4ビット，8ビットとそれぞれの過程と経験を経て，より優れた能力が追求されて出現したのが16ビット・マイクロプロセッサであり，私たち人間の限りなき挑戦の成果のたまものであると思います．メモリ容量の増大，命令数の増大，命令実行スピードの速さなど，その技術進歩は急激であり，取り扱う私たちがその技術の吸収に追従しえないほどのすさまじさであります．

16ビット・マイクロプロセッサは，ハードウェア面においては，ミニコンの領域を越えて汎用小型・中型計算機に匹敵する能力を備えているといっても過言でないほどの強力な素子といえましょう．

また，だれの目にも明らかに映るのは，外形の大きさに比べその性能が驚異的なことであり，従来の小規模集積回路（SSI）や中規模集積回路（MSI）に比べ，集積度やプロセス技術が数倍に飛躍していることです．私たち人間の限りなき技術への挑戦と，それをかちえてゆく技術者に賞賛と敬意を払います．

現在，第2世代の16ビット・マイクロプロセッサの時代に突入したわけですが，8ビット・マイクロプロセッサにとって代わって，その隆盛を見るためには，よりいっそうのソフトウェアの充実が課題でありましょう．また，8ビットから16ビットへと移行するにつれて，デバイス価格もどんどん低減してゆき，それが引金となって種々の分野へと需要が広がってゆくと思われます．また，第3世代の32ビット・マイクロプロセッサの時代も目前に控えており，より充実したソフトウェアの開発は，今後よりいっそう要求されるものと思われます．

以上のような背景で，本書は16ビット・マイクロプロセッサ，MC 68000とはどうゆうものかを理解していただくことに重点を置いて執筆いたしました．また，紙数の都合上，68000とのインタフェース関係，プログラミング関係，アプリケーション関係の執筆が不足しておりますが，別の機会に紹介できればと考えております．

は　し　が　き

各引用資料として，本書の末尾に参考資料を掲載しておきましたので，読者諸氏がMC 68000の検討や導入に際してご参照くだされば，よりいっそうのご理解を得ていただけるものと信じます．

なお，本書執筆にあたり，ご協力いただいた日本モトローラ(株)伊南恒志氏，および末尾の参照資料，写真などの引用のご承諾をいただいた日本モトローラ(株)広報課，マーケッティング課のご協力に深く感謝いたします．また，本書の編集，出版に際して，オーム社出版部の方々に大変なお世話をいただき，この出版ができたことに厚くお礼申し上げます．

昭和58年8月

著者しるす

目　　　次

6. 例　外　処　理

7. 命令セット

8. システムの構成

9.　周辺ファミリ・チップ

10.　MC 68000 開発装置

11.　開発ソフトウェア/リアルタイム・モニタ

12.　MC 68000 の将来の発展方向

付　　　録

1. 16ビット・マイクロプロセッサ入門

16ビット・マイクロプロセッサが作られるまでの背景と，モトローラ社のMPUの推移を説明し，8ビットMPUと16ビットMPUの比較および16ビットの特長を記しています．また，それぞれの開発支援装置の推移も説明しています．

1·1 MC 68000 の誕生まで

マイクロコンピュータは，約10年前に発表されて以来，各方面のニーズにこたえるべく，技術革新とともに急速な進歩と発展を遂げてきています．モトローラ社が8ビット・マイクロプロセッサMC 6800を発売した年は1974年であり，16ビット・マイクロプロセッサMC 68000が誕生するまでには，その後5年の年月が費やされ，1979年に発売されて今日に至っております．また他社においては，爆発的人気の出たインテルの8080が1973年に発売され，そしてザイログのZ80が1976年に次々と発売され，今日の8ビット・マイクロプロセッサの隆盛を見るに至っております．

年月の経過とともに用途，方法も年々向上し，応用範囲も広がり，より高速な数値演算やマルチプログラミングによるメモリ管理方法，膨大なメモリを必要とするデータ処理方法など，より高度な使用方法や要求が今日の8ビット・マイクロプロセッサの能力を超えるものとなり，16ビット・マイクロプロセッサが使用されるようになった背景です．また製造面についても，回路技術の進歩，エッチング方法の進歩，プリンティング方法の進歩など大きな技術的躍進がなされたことです（1·2節「8ビットMPUから16ビットMPUへ」参照）．

モトローラのMPUの超LSI回路技術は，図1·1に示すように，発展方向は二つの方向をたどりました．機能面が強化された方向として，MC 6808，MC 6802，MC 6801などのマイクロプロセッサがあり，効率面が強化された方向としては，MC 68 A 00，MC 68 B 00，MC 6809などのマイクロプロセッサがあります．

68000は，このようなMC 6800系でのアーキテクチャの基盤と集積度技術の向上，そしてプロセス技術の向上などの蓄積と統合によって生まれてきたのです．その応用分野は，従来の特殊分野から脱皮し，OA関係，ロボット関係の工場や各機器の自動化，民主機器関係の導入へと広がり，16ビット・マイクロプロセッサ主流の時代に突入しようとしています．また一部，第3世代の32ビットMPUの要望もすでに出ており，モトローラ社の上位に位置する仮想MPU，MC 68020，

32ビットMPUも1983年に発売が予定されています.

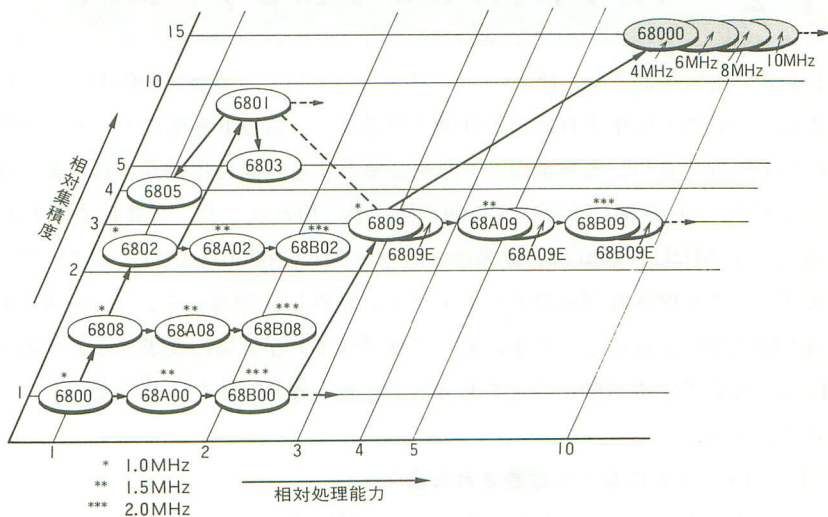



図 1·1　モトローラ・マイクロプロセッサの歩み

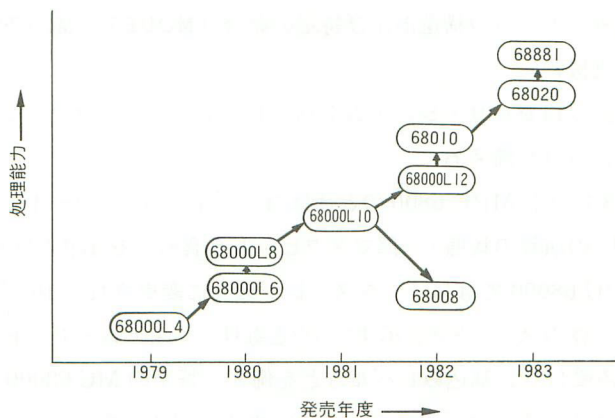



図 1·2　16/32ビット・マイクロプロセッサ相関図

1・2　8ビットMPUから16ビットMPUへ

1・1章で述べたように，16ビット・マイクロプロセッサは，時代のニーズにこたえるべく必然的に生まれたことは明らかです．ロボット制御に用いるより精度の高い座標軸計算や，大容量のメモリを必要とする画像処理とか，日本語処理に便利な16ビットのMPUが，本格的に検討や応用がなされ，普及しています．

16ビットMPU 68000は，従来の8ビットMPU 6800とのコンパチビリティおよび一貫性を保ちながらのアーキテクチャや超LSI回路技術，プロセス技術の改良と開発がなされたことです．また，サプライヤが市場の要求に応じうる十分な供給と適応できる価格が可能であったことも，普及しえた大きな要因の一つといえましょう．

〔1〕 16ビットになって改善された点

（1） 多数のレジスタとスタック，広いアドレス・レンジ，高級言語指向命令（LINK，UNLK，CHKなど）による高級言語の支援．

（2） オペレーティング・システム（OS）：特権命令，メモリ管理，ベクタ多重レベル割込み，トラップ構造および特定の命令（MOUEP，MOUEM，TRAPなど）による支援．

（3） マルチプロセッサ・システム：ハードウェア・インタロックとソフトウェア・インタロックを備える．

フラグは，8ビットMPU 6800のときはコンディション・コード・レジスタといわれ，MPUの演算の状態（一部マスクビットを含む）を示すだけであったが，16ビットMPU 68000ではステータス・レジスタに変更されて16ビットに拡張され，MPU全体のステータスを示すフラグとなり，システム・モード，ユーザ・モード，割込み受付け，割込みレベルなどを備え，将来のMC 68000のファミリの拡張に備えて十分広いスペースをとっています（図2・4参照）．

〔2〕 命令シーケンスと処理速度の代表的改善点　クイック・イミディエート・アドレッシング・モードを用いたシングル・ワード・オペレーションの加算や減算命令は，データ・レジスタとメモリの処理速度を速めました．MOVEQ

(move quick) 命令では，シングル・ワード・オペレーションで符号付きバイト・データを任意のレジスタにロードできます．また，DBcc (decrement and branch) 命令ループは，オペレーションの処理速度を速めるために，一つの命令でコンディションのテストを行い，デクリメント分岐ができます．また，コード減らしや演算オペレーションの性能を高める命令として，MULS (signed multiply), MULU (unsigned multiply), DIUS (sined divide), DIVU (unsigned divide) があり，演算オペレーション命令としてABCD, SBCDなどがあります．

ペリフェラル・コントロールとのインタフェースについても，MC 68000と68系8ビットMPUに開発された周辺デバイスが直接インタフェースできるようになっています．シグナルE (enable), $\overline{\text{VMA}}$ (valid memory adress), $\overline{\text{VPA}}$ (valid peripheral adress) などが設けられています（図2·5，図4·12参照）．

表 1·1 16ビット・マイクロプロセッサ製造会社

タイプ	海　外	日　本
MC 68000	モトローラ モステック，フィリップス ロックウェル，シグネテックス	日　立
I 8086	インテル AMD ハリス シーメンス	日本電気 富士通 三菱電機 東　芝
Z 8000	ザイログ AMD	シャープ 東　芝
TMS 9995	テキサス・インスツルメント	—
NS 16000	ナショナル・セミコンダクタ	—
F 9445	フェアチャイルド	—

表 1·2 8ビットMPUと16ビットMPU比較

区　分	6800	68000
集積度	NMOS	HMOS
エッチング	ケミカル・エッチング	ドライ・プラズマ・エッチング
プリンティング	ダイレクト・コンタクト・プリンティング	プロジェクション・プリンティング
ブロック・ゲート面積	4128 μm^2	1852 μm^2
速度電力積	4ピコジュール	1ピコジュール

〔3〕 **開発支援装置**　　開発支援ツールについても，図 1·3 に示すように 8 ビット MPU から 16 ビット MPU に移るとともに各装置が出ております．8 ビット MPU 用エクササイザ (EXOR ciser) に始まり，エクサセット (EXOR set)，16 ビット MPU 用のエクササマクス (EXOR macs)，そしてボックス・コンピュータ (VMC 6812) と MC 68010 VM プロセッサ搭載のマイクロコンピュータシステム (VME/10) のような推移と種類の拡大が行われてきています．

なお，今後とも引き続きマイクロプロセッサの発展とともに，各種各様の多くの開発支援装置が生まれてくると思います．

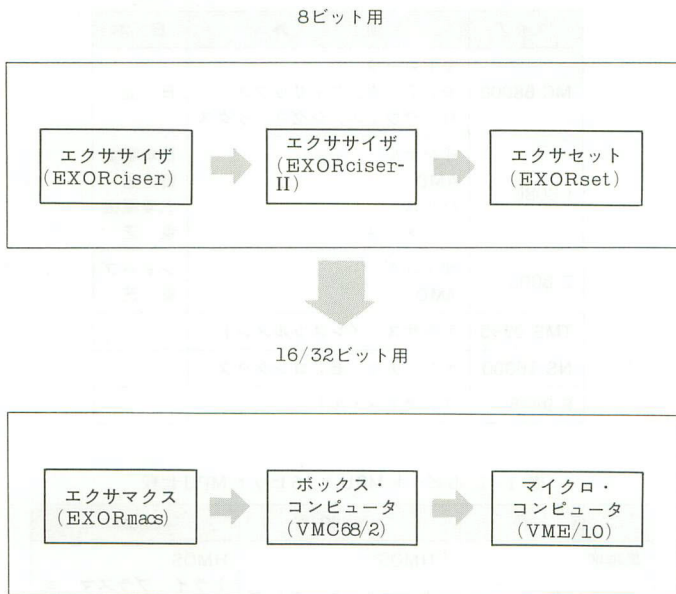



図 1·3　開発支援装置の移行

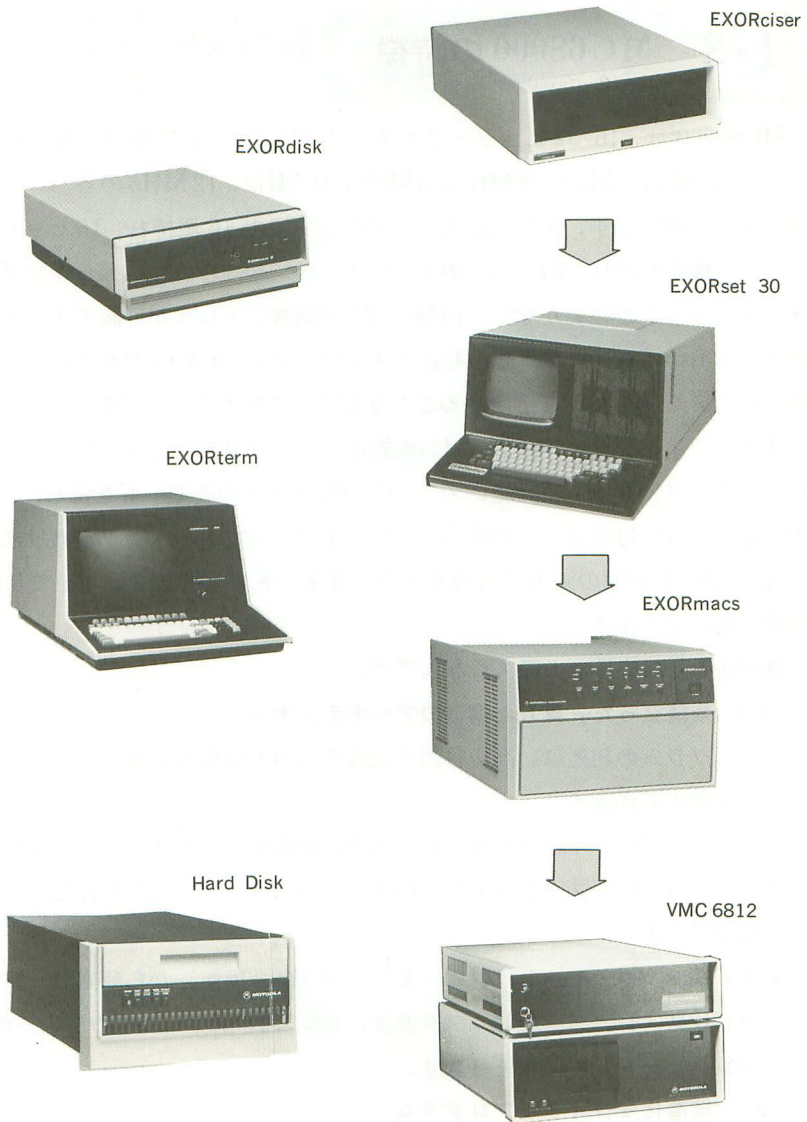



図 1・4　モトローラ MPU 開発支援装置

1・3　MC68000の特徴

MC68000は，16/32ビット・マイクロプロセッサとして設計されており，クロック周波数は4MHz，6MHz，8MHz，10MHz，12MHzの各バージョンが用意されております．また，32ビットのレジスタを16個持ち，16ビットのデータ・バス構成をとり，24ビットのアドレス・バスを持ちます．ページングやセグメンテーションの必要がなく，16Mバイトの直接アドレスが可能です．アーキテクチャにおいても，一貫思想のもとにメモリ・マップドI/Oが忠実に受け継がれており，メモリとI/Oを意識することなく同等に扱えることです．

オペレーティング・システム用の機能として，ソフトウェア・トラップ命令，ハードウェア・トラップ機能，スーパバイザ/ユーザの各モードを持ち，7レベルの外部割込みに対応できる192のユーザ・インタラプト・ベクタがあります．バスは，同期/非同期の両方式が使用でき，従来の8ビットMPUファミリのLSIを直接接続できます．

68000の主な特長は次のような点です．

（1）　系統立った一貫した構造のアーキテクチャ

- プログラムの記述は，高級言語で記述するのと同様な容易さでアセンブラ言語が使用できます．
- フレキシビリティと能力を大幅に高め，整数データのオペランド操作におけるデータ・レジスタとメモリ・ロケーションは，ソースにも行先にも自由に指定できます．
- アドレシング・モードは，ニーモニック（命令操作）から独立し，すべてのアドレス・レジスタをレジスタ直接，レジスタ間接，インデックス・アドレッシング・モードで使用できます．

（2）　構造化モジュラ・プログラム

- ブロック構造高級言語（構造化アセンブラやPASCALなど）が使用できます．
- 総計255のベクタ・ロケーションを持ち，これらの割込みをハードウェア・

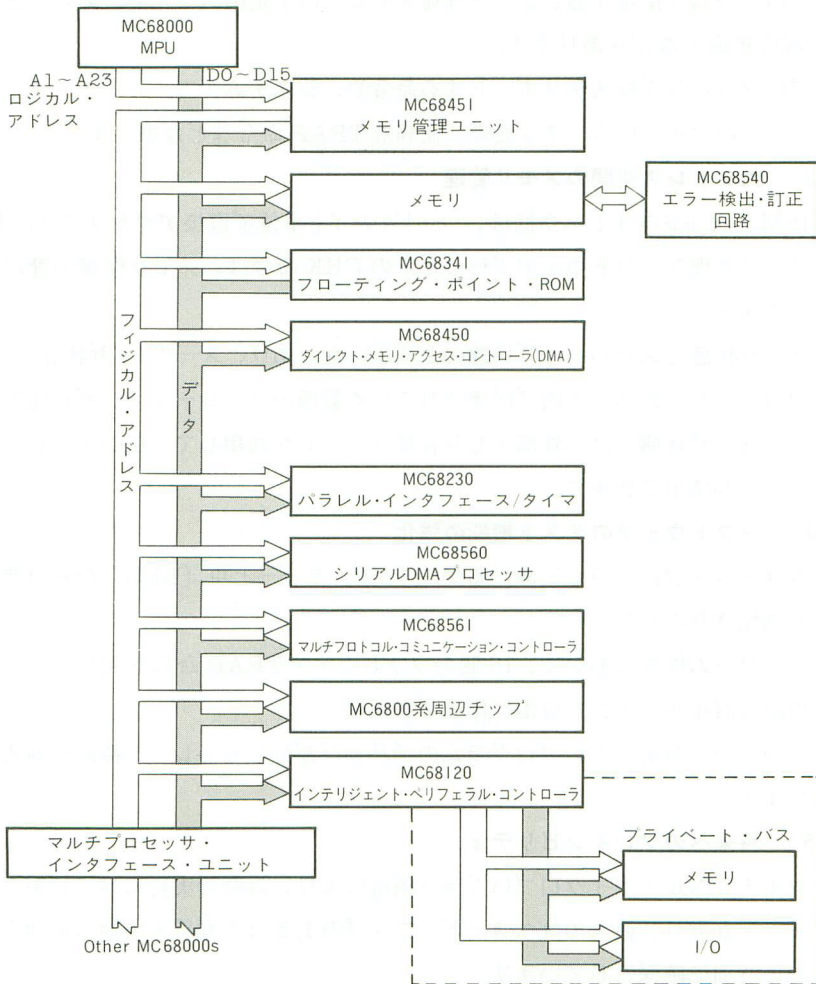



図 1・5　ローカル・バスの拡張

トラップ，ソフトウェア・トラップに割り当てています．

- サブルーチン・コールのオーバヘッドを減少させるLINK，UNLK命令，プログラムが指定するレジスタ群をアドレスの指定のみで一度に転送できるMOVE命令などがあります．
- プログラミング技法をサポートする命令や，システム・コール・ルーチンやユーザのマイクロルーチン処理に便利なTRAP命令などがあります．

(3)　大アドレス空間のメモリ管理

- 16Mバイトのアドレス空間は，ワードやバイト単位で直接アクセスできます．
- メモリ管理をソフトウェアで行うときのCHK命令は，メモリ保護や管理ができます．
- ユーザ状態とスーパバイザ状態の2種の区別があり，スーパバイザ状態では，プロセッサ・システム内で保護されている数種のオペレーションが可能であり，ユーザ状態では，外部メモリ管理ユニットを利用して，広いアドレス・レンジが使用できます．

(4)　ソフトウェアのテスト機能の強化

- システム・プログラムにおけるプログラム・エラーと虫（BUG）の検出機能が強化されました．
- プログラム作成において，16個のソフトウェアTRAP命令を用いてエラー検出と訂正ルーチンが自由に作成できます．
- 一つ一つの命令ごとにプログラムのデバッグが行えるトレース機能を備えています．

(5)　将来へのフレキシビリティ

- マルチレベル・マイクロプログラム構造により，命令の実行にはフレキシビリティがあり，命令のオペコード・マップの1/8以上が将来の命令追加のために特別に確保されています．
- ユーザは，現在の命令セットにない命令をトラップ命令とエミュレータ・トラップを用いて作成，実行が可能です．

さらに現在では，MC68000の上位機種も続々と発表される予定です．バーチャルメモリ，バーチャルI/Oを強力にサポートするMC68010，32ビットMPUの

システム・インタラプト

MC68000の割込みレベルは7レベルありますが，“スピードマスタ68K”では，そのうちの6個を使用しています．

下表に，割込みレベルの各デバイスへの割合を示します．

表 1・3　割込みレベル

レベル	デバイス
1	未使用
2	PI/Tタイマ
3	PI/Tパラレル・ポート
4	M6800インタフェース*
5	ACIA 1*
6	ACIA 2*
7	アボート・スイッチ*

＊　オートベクタ

メモリ・マップ

表 1・4　メモリ・マップ

	アドレス	資源
ROM領域	S000000-S000007*	ROM/EPROM［イニシャル・スタック・ポインタ イニシャル・プログラム・カウンタ］
RAM領域	S000008-S0007FF	システムRAM［作業領域として使用するシステム・カム］
	S000800-S007FFF	ユーザRAM
ROMファームウェア	S008000-S00BFFF*	ROM/EPROM
(斜線)	S00C000-S00FFFF	使用していない
I/Oデバイス 64Kバイト・ページ	S010000-S01003F	PI/T（下位バイト）
	S010040-S010043	ACIA2（下位バイト） ACIA1（上位バイト）
	S010044-S01007F	拡張用ACIA
	S010080-S0100BF	拡張用PI/T
	S0100C0-S0100FF	拡張用ACIA
	S01100-S01FFFF	拡張用ACIAとPI/T

0000 0001 xxxx xxxx x0xx xxx0 PI/T
0000 0001 xxxx xxxx x0xx xxx1 使用せず

0000 0001 xxxx xxxx x1xx xxx0
ACIA2
0000 0001 xxxx xxxx x1xx xxx1
ACIA1

	アドレス	資源
(斜線)	S020000-S02FFFF	使用していない
特別なデコード信号	S030000-S03FFFF	6800 64K byteページ［周辺デバイスのインタフェース用］
(斜線)	S040000-SFFFFFF	使用していない

＊　リードのみ

図 1・6　スピード・マスタ68Kの割込みレベルとアドレス・デコード

MC68020，浮動小数点演算用コ・プロセッサなどがあります．また，内部がMC68000と完全にコンパチブルで，外部バスのみ8ビット構成で，中規模システムのハードウェア・コスト低減とソフトウェア向上の用途としてMC68008などがあります（12·3節参照）．

16ビット・マイクロプロセッサMC68000の入門用や学習用として，安価な単一ボード・コンピュータ“**SPEED MASTER 68K**”を紹介します．

図1·7にスピード・マスタの実線図を示します．また，図1·8にブロック図を示します．

ファームウェア：ファームウェアはプログラム作成とオペレーションを制御し，2個の8Kバイト ROMに格納されています．そして，ユーザにデバッグ/モニタ機能，プログラム入力，アセンブラ，逆アセンブラ，I/O機能を提供します．

デバッグ/モニタ：他のモトローラ製品と互換性を持たせるため，用意されているコマンドはすべてMACSbug，VERSAbugのコマンド形式と同一です．デバッグ機能には，メモリ表示/変更，レジスタ表示/変更，プログラム・トレース，ブレーク・ポイント，プログラム実行，データ変換，トランスパーレント・モードなどがあります．

アセンブラ/逆アセンブラ：ソース・プログラムが直接ファイル化されないダイナミックなアセンブラ，エディタ・プログラムです．各命令は1行入力するごとに適切な機械語に翻訳され，メモリへ格納され，翻訳された機械語を逆アセンブルし，ニーモニックとオペランドを表示します．

I/O機能：端末を介してのユーザ・インタフェースのサポートに加え，ファームウェアは他のI/O機能もサポートします．ホスト・コンピュータからのオブジェクト・プログラムの転送やオーディオ・カセットへのオブジェクト・プログラムの格納と取出しやプリンタの制御などを支援します．

図 1・7　スピード・マスタ 68K 図

MC68000
ADDRESS
DATA
16Kbyte ROM
8MHz CLOCK
32Kbyte RAM
MC68230
MC6850
MC6850
BUFFER
BUFFER
MC14411 BAUD RATE GENERATOR
パラレル ポート
RS-232C ポート 2
RS-232C ポート 1
Printer or I/O
Cassette Recorder
Host Computer
Terminal

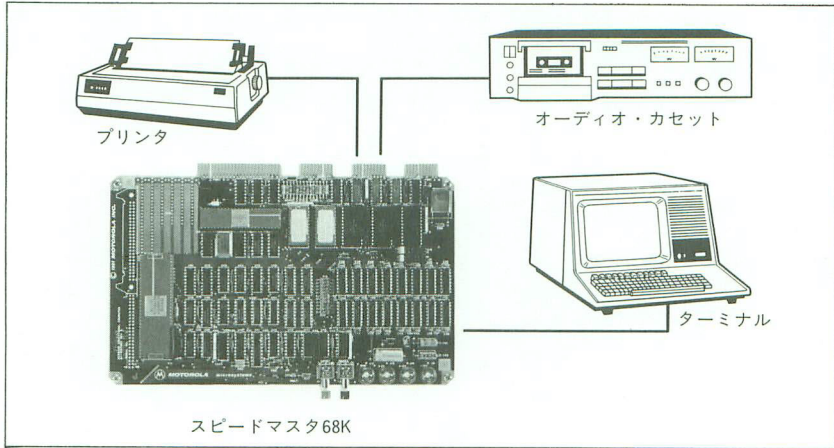
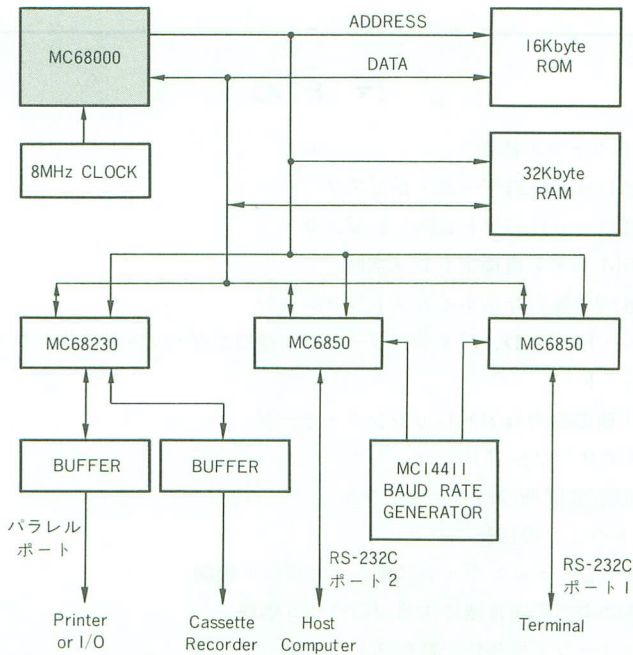



図 1・8　スピード・マスタ 68K ブロック図

〔マ　ト　メ〕

1．ハードウェアの特長
　1）32ビット長のデータ・レジスタ
　2）32ビット長のアドレス・レジスタ
　3）16Mバイト直接アドレス空間
　4）56種の強力な基本インストラクション
　5）ビット，BCD，バイト，ワードおよびロング・ワードのデータ・タイプをサポート
　6）14種の強力なアドレッシング・モード
　7）メモリ・マップド I/O
　8）同期/非同期のバス・サイクル
2．ソフトウェアの特長
　1）オペレーティング・システム・サポート機能
　2）高効率で高級言語によるプログラム処理
　3）クリーンで拡張性に富むストラクチャ

2. MC68000の構成

MC68000のピン数やピン端子名の役割と機能について一覧表で説明し，内部におけるレジスタの数やレジスタ構成を図示し，おのおののレジスタの役割についての概要を説明します．

2・1 ピ ン 構 成

MC68000は，64ピンのDIP（Dual Inline Package）に収められたHMOS（High Density Short Channel MOS）のVLSのICです．

MC68000のピン端子番号と信号の関係は図2・1に示します．また，シグナル名と機能は表2・1に示します．

各シグナルの詳細な機能については，第4章の入/出力の構成を参照してください．V_{cc}電源は，+5V単一電源（最小2.4V）で動作します．また，MPU内部のレジスタはダイナミックRAMと同様な動作になっていますので，データの取扱いとクロック周波数には配慮が必要です．

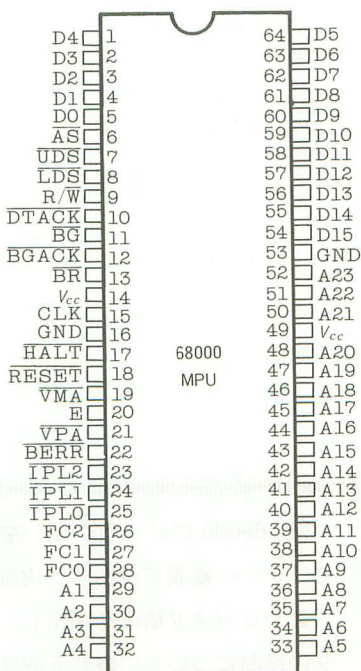



図 2・1 MPUとピン番号

表 2・1　ピン番号と信号の関係

ピン番号	信号名	機能	形式
14	Vcc	電源	入力
49	Vcc		
16	GRD	グランド	入力
53	GRD		
15	CLK	クロック	入力
5	D0	データ・ライン	入力/出力 (双方向3ステート)
4	D1		
3	D2		
2	D3		
1	D4		
64	D5		
63	D6		
62	D7		
61	D8		
60	D9		
59	D10		
58	D11		
57	D12		
56	D13		
55	D14		
54	D15		
29	A1	アドレス・ライン	出力 (単方向3ステート)
30	A2		
31	A3		
32	A4		
33	A5		
34	A6		
35	A7		
36	A8		
37	A9		
38	A10		
39	A11		
40	A12		
41	A13		
42	A14		
43	A15		
44	A16		
45	A17		
46	A18		
47	A19		
48	A20		
50	A21		
51	A22		
52	A23		
6	$\overline{AS}$	アドレス・ストローブ	出力
7	$\overline{UDS}$	アッパ・データ・ストローブ	出力
8	$\overline{LDS}$	ローア・データ・ストローブ	出力
9	$R/\overline{W}$	リード/ライト	出力
10	$\overline{DTACK}$	データ・トランスファ・アクノリッジ	入力
11	$\overline{BG}$	バス・グラント	出力
12	$\overline{BGACK}$	バス・グラント・アクノリッジ	入力
13	$\overline{BR}$	バス・リクエスト	入力
17	$\overline{HALT}$	ホルト	入/出力
18	$\overline{RESET}$	リセット	入/出力
19	$\overline{VMA}$	バリッド・メモリ・アクセス	出力
20	E	イネーブル	出力
21	$\overline{VPA}$	バリッド・ペリフェラル・アドレス	入力
22	$\overline{BERR}$	バス・エラー	入力
23	$\overline{IPL2}$	インタラプト・コントロール	入力
24	$\overline{IPL1}$		
25	$\overline{IPL0}$		
26	FC2	プロセッサ・ステータス	出力
27	FC1		
28	FC0		

2・2 レジスタの構成

MC68000 の特徴である 32 ビット長の合計 15 個の汎用レジスタと 4 個のコントロール・レジスタは，仕様を十分に満たし，魅力あるチップといえましょう．また，演算レジスタとアドレス・レジスタが分離していることも特徴の一つといえましょう．

68000 のレジスタの構成を機能別に分類しますと，次の 5 種類に分けられます．1）データ・レジスタ，2）アドレス・レジスタ，3）スタック・ポインタ，4）プログラム・カウンタ，5）ステータス・レジスタ．

〔1〕 **データ・レジスタ**　D0～D7 までの 8 個のデータ・レジスタを備え，1，8，16，32 ビットのおのおののデータ・オペランドをサポートします．また，演算や転送などの各命令オペランドサイズは，B＝バイト（8 ビット），W＝ワード（16 ビット），L＝ロング・ワード（32 ビット）があります．そして，バイトやワードの下位ビットをソース・オペランドする場合や行先オペランドの場合は，該当部分が変わり，残り上位ビットは使われません．

〔2〕 **アドレス・レジスタ**　A0～A6 までの 7 個のアドレス・レジスタを備え，アクティブ・スタック・ポインタとともに，32 ビットのアドレス・オペランドをサポートします．A0～A6 は汎用インデックス・レジスタとして用い，アドレス・モードとともに全アドレス空間を柔軟に用いることができます．アドレス・レジスタのアクセスは，ワード（16 ビット）かロング・ワード（32 ビット）のどちらかを用います．したがって，バイト・サイズのオペランドは使用できませんので注意してください．

〔3〕 **スタック・ポインタ**　A7 のスタック・ポインタは，16 ビットのユーザ・スタック・ポインタと 16 ビットのスーパバイザ・スタック・ポインタの 2 種類を備え，どちらか一方のステートで動作します．また，スーパバイザ状態でシステム・スタック・ポインタとして働き，トラップや割込み時のプログラム・カウンタとステータス・レジスタのスタックにも有効です．

31 16 15 8 7 0

データ・レジスタ		
0	D0	データ・レジスタ（8個）
1	D1	
2	D2	
3	D3	
4	D4	
5	D5	
6	D6	
7	D7	

31 24 23 16 15 0

アドレス・レジスタ		
0	A0	アドレス・レジスタ（7個）
1	A1	
2	A2	
3	A3	
4	A4	
5	A5	
6	A6	

ユーザ・スタック・ポインタ（USP）	A7	スタック・ポインタ（2個）
スーパバイザ・スタック・ポインタ（SSP）		

31 0

	PC	プログラム・カウンタ（1個）

システム・フラグ　ユーザ・フラグ

15 13 8 4 0

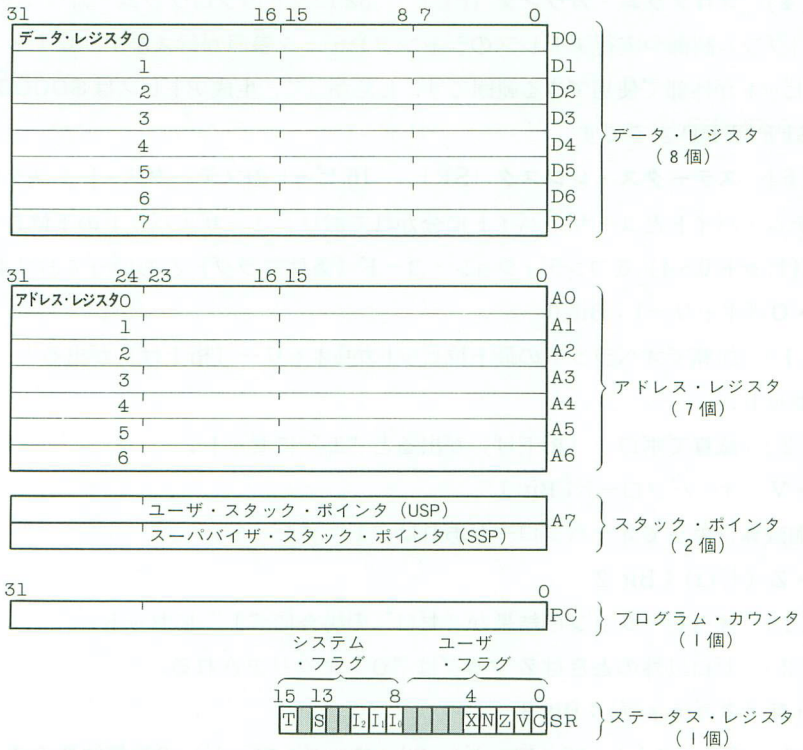
T		S			I2	I1	I0				X	N	Z	V	C	SR

ステータス・レジスタ（1個）

図 2・2　68000のレジスタ構成

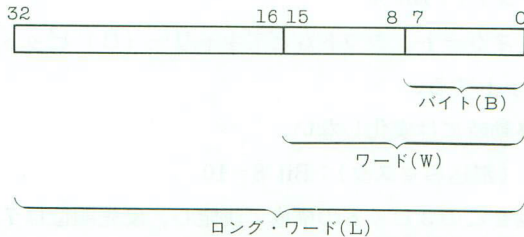



図 2・3　レジスタのデータ処理サイズ

［4］ **プログラム・カウンタ**（PC）　32 ビットのプログラム・カウンタは，プログラム制御や実行アドレスのジャンプやデータ参照が行えます．32 ビット中 24 ビットが外部で使用できる範囲です．したがって，生成アドレスは＄000000～＄FFFFFF までです．

［5］ **ステータス・レジスタ**（SR）　16 ビットのステータス・レジスタはシステム・バイトとユーザ・バイトに分かれており，ユーザ・バイトの下位 5 ビット（ビット 0～4）をコンディション・コード（条件フラグ）ともいっております．

・C（キャリー）：Bit 0

（1） 加算でオペランドの最上位ビットからキャリー（桁上げ）が出ると“1”にセット．

（2） 減算でボロー（桁下げ）が出ると“1”にセット．

・V（オーバフロー）：Bit 1

加減算によってオーバフローがあれば“1”にセット．

・Z（ゼロ）：Bit 2

（1） オペレーションの結果が“ゼロ”の場合に“1”にセット．

（2） ゼロ以外のときは Z フラグは“0”にクリアされる．

・N（ネガティブ）：Bit 3

（1） 8（バイト），16（ワード），32（ロング・ワード）の演算結果の最上位ビットが“1”（負）であった場合に“1”にセット．

（2） それ以外の場合，N フラグは“0”にクリアされる．

・X（エクステンド）：Bit 4

（1） 加減算，ネゲート，シフトなどでキャリー（C）ビットと同じように変化し，“1”にセットする．

（2） データ移動時には変化しない．

・I_0，I_1，I_2（割込みマスク）：Bit 8～10

割込み優先順位をこの 3 ビットの構成で決定し，優先順位は 7 が最も高く，1 が最下位です（表 2·2 参照）．

・S（スーパバイザ）：Bit 13

（1） S＝1 のときスーパバイザ・ステート（特権状態で，上位にある）．

（2）　S＝0 のときユーザ・ステート（特権状態でも下位に位置します）.

・T（トレース・モード）：Bit 15

T＝1 のときデバッグ機能によりシングル・ステップでテスト中のプログラムの実行をモニタできます.

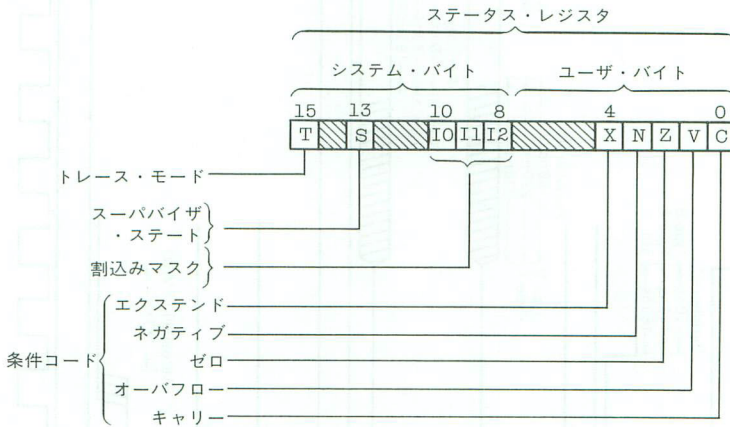



図 2・4　ステータス・レジスタ

表 2・2　割込みマスクとオートベクタの関係

割込み マスク	I_2 I_1 I_0	割込み 優先度	割込み内容	ベクタ 番号	ベクタ アドレス (HEX)
7	1 1 1	1位	LEVEL7 INTRRUPT AUTO VECTOR	31	07C
6	1 1 0	2位	LEVEL6 〃	30	078
5	1 0 1	3位	LEVEL5 〃	29	074
4	1 0 0	4位	LEVEL4 〃	28	070
3	0 1 1	5位	LEVEL3 〃	27	06C
2	0 1 0	6位	LEVEL2 〃	26	068
1	0 0 1	7位	LEVEL1 〃	25	064

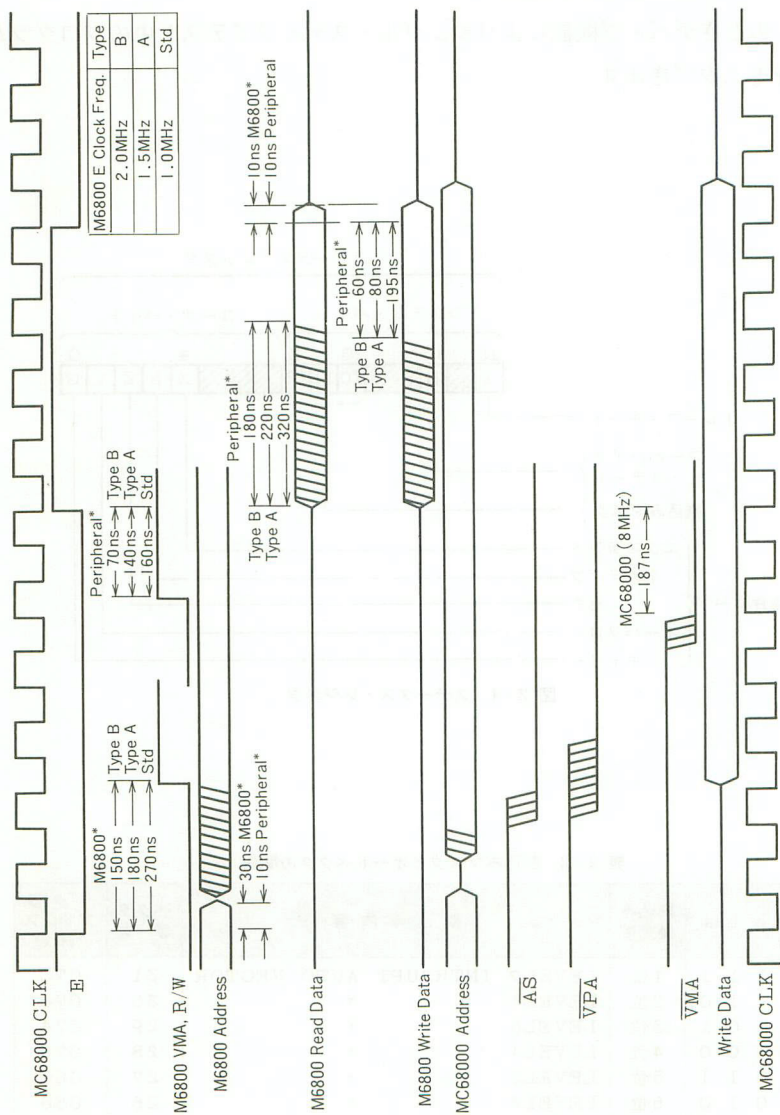



図 2・5 68000と6800周辺デバイスをインタフェースしたときのタイミング

3. メモリの構成

MC68000 が持つ 16M バイトのメモリ構成や，メモリの区分，割付けについて述べ，そのメモリ内で扱われるデータ構成とデータ転送について説明します．

また，スタックやキューの概念を述べ，実際の使用について考察しました．

3・1 メモリの構成

MC68000は，A1～A23までの23本のアドレス出力ピンと，プロセッサ内部のA0信号により，2^{24}バイト＝16Mバイトのアドレス空間を直接アクセスすることができます.

この16Mバイトのアドレス空間は，連続した8Mワードの構成（図3・1参照）か，あるいは4Mのロング・ワード構成をとることができます.

MC68000は，他のプロセッサに見られるI/Oなどの命令はなく，メモリ・マップドI/O方式（I/Oなどのデバイスとのデータ・アクセスにおいて，I/Oを特に意識することなく，I/Oデバイスをメモリ内に割り付け，見かけ上メモリtoメモリの転送として処理する方式）をとり，ページング・セグメンテーション（アドレス空間を，ある基準に基づいて区分する方法）を用いないために，直接16Mバイトのメモリ構成のどこにでもI/Oアドレス領域を複数に割り付けることが可能です.

16Mバイトのアドレス空間のうち，16進HEXアドレスで000～0FFまでが，システムの例外処理ベクタ・アドレス・テーブルに割り付けられ，また，100～3FFまでがユーザの割込みベクタ・アドレス・テーブルに割り当てられています．したがって，ユーザに開放されているアドレスは000400～FFFFFFとなります（図3・1参照）.

MC68000は，前述した3本のファンクション・コード出力信号を四つに分類します．よって，ファンクション・コード出力信号を外部メモリのチップ・イネーブルとして用いることで，最大64Mバイトのメモリ領域をアクセスすることができます.

また，FC2出力信号を図3・2で示すようにチップ・イネーブルとして用いる場合，メモリ・プロテクションをすることが可能です.

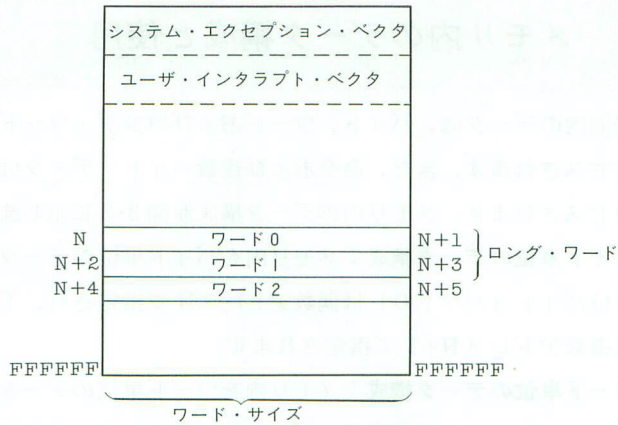



図 3・1 メモリ構成

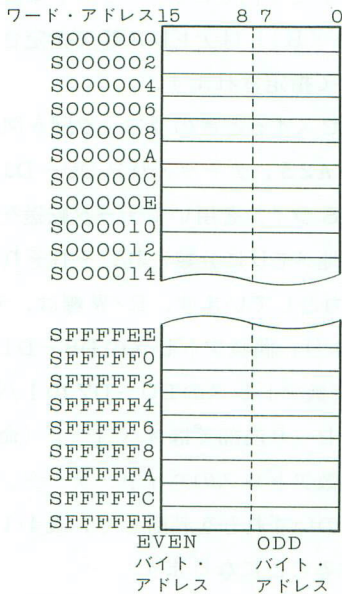



図 3・2 メモリ・マップ

3•2 メモリ内のデータ構成と使用

メモリ空間内のデータは，バイト，ワードおよびロング・ワードのデータ・サイズでアクセスされます．また，命令および複数バイト・データは，ワード単位でのみアクセスされます．メモリ内のデータ構成を図3·3に示します．

（1） **バイト単位のデータ構成**：メモリ内をバイト単位のデータ系列として扱う場合，上位バイト（バイト0）は偶数アドレスNで指定され，下位バイト（バイト1）は奇数アドレスN+1で指定されます．

（2） **ワード単位のデータ構成**：メモリ内をワード単位のデータ系列として扱う場合，ワード・データは必ず偶数アドレスによりアクセスされます．例えば，ワード0のデータはアドレスNにより指定され，次のワード・データ；ワード1はアドレスN+2により指定されます．

（3） **ロング・ワード単位のデータ構成**：メモリ内をロング・ワード単位のデータ系列として扱う場合，上位ワード（ワード（H））はアドレスNで指定され，下位ワード（ワード（L））はアドレスN+1で指定されます．

MC68000がバイト単位のデータをアクセスするときのブロック図を図3·4に示します．このとき，アドレス・バスA1～A23，データ・バスD0～D15のほかに，制御ラインとしてR/$\overline{\text{W}}$，$\overline{\text{LDS}}$，$\overline{\text{UDS}}$ラインを用いてデータ転送を実行します．メモリは，偶数番地メモリと奇数番地メモリに分類され，それぞれ$\overline{\text{LDS}}$，$\overline{\text{UDS}}$をアクティブ“LOW”のセレクタ入力としています．R/W線は，データ・バス上のデータの方向を決め，$\overline{\text{UDS}}$線により，偶数アドレスのD8～D15の1バイト転送を可能にし，$\overline{\text{LDS}}$線により，奇数アドレスのD0～D7の1バイト転送を可能にしています．したがって，プロセッサ内部ではオペコード（命令）を読み取り，オペランド（転送データ）が偶数アドレスのバイト・データ，奇数アドレスのバイト・データ，ワード・データのいずれかを判断して，表4·1に示された論理レベルを$\overline{\text{UDS}}$，$\overline{\text{LDS}}$線に出力することになります．

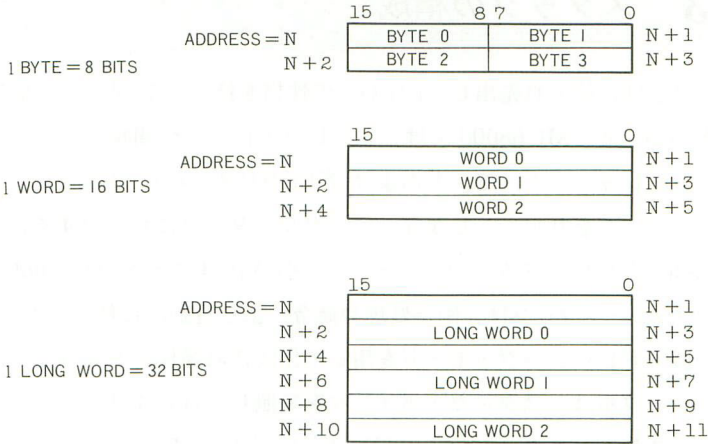



図 3・3 データ・メモリ・フォーマット

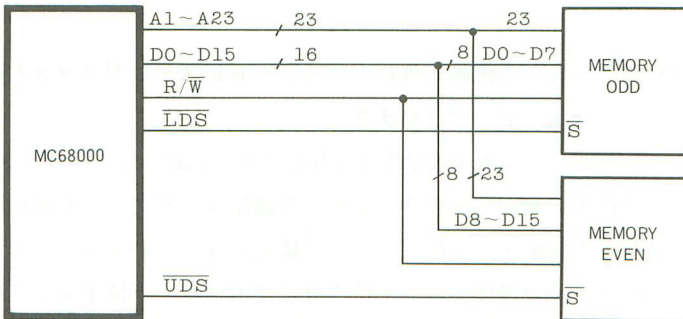



図 3・4 バイト・アドレッシング

3・3 スタックの構成

スタックとは，後入れ先出し（LIFO）の性格を持った配列データ構造のメモリ領域をいいます．MC68000では，アドレス・レジスタ間接，ポストインクリメント・アドレッシング・モードおよびプリデクリメント・アドレッシング・モードによりスタックをサポートします．データをスタックにセーブすることをプッシュ（push）といい，スタックからデータを引き出すことをプル（pull）といいます．システム・スタックは，例外処理や命令により自動的に使用され，ユーザ・スタックはアドレッシング・モードを用いて作成され操作されます．スタックへのプッシュ・プルは，スタック・ポインタを参照して行います．

［1］ **システム・スタック**　アドレス・レジスタA7が，システム・スタック・ポインタとして使用されます．また，ステータス・レジスタの“S”ビットの状態に応じ，スーパバイザ・スタック・ポインタ（SSP）あるいはユーザ・スタック・ポインタ（USP）として役目を果たします．すなわち，スーパバイザ・モードではSSPとしてのみ働き，ユーザ・モードではUSPとしてのみ働くことを示し，一方のモードにおいて双方のスタック・ポインタのアクセスを防いでいます．

トラップや割込みなどの例外処理において，アドレス・レジスタA7はスーパバイザ・スタック領域へセーブされます．

ユーザ・モードで用いられるRTS命令などにより，A7はユーザ・スタック・ポインタとして自由に使用できます．スタック操作は一般にワード単位で行い，システム・スタック内のデータ配列をワード構成に保っています．このため，バイト・データのスタック操作では，上位半分が使用され，下位半分は使用されません．

［2］ **ユーザ・スタック**　システム・スタックが，例外処理やRTS命令などにより自動的にスタック操作を行うのに対し，ユーザ・スタックは，アドレス・レジスタ（A0～A6）をスタック・ポインタとして利用し，ポストインクリメントおよびプリデクリメントのアドレス・レジスタ間接アドレッシング・モードを

使用することにより作成され操作されます.

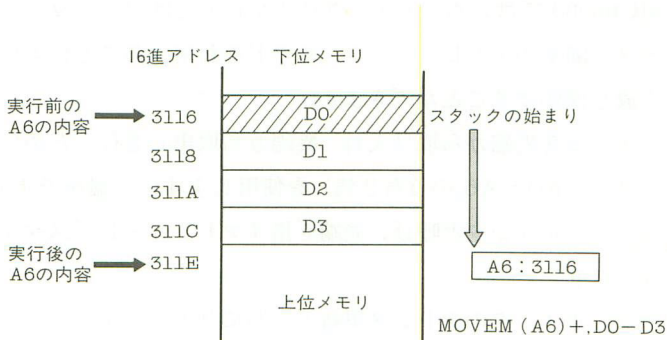



図 3・5 ポストインクリメントを用いたプッシュ操作

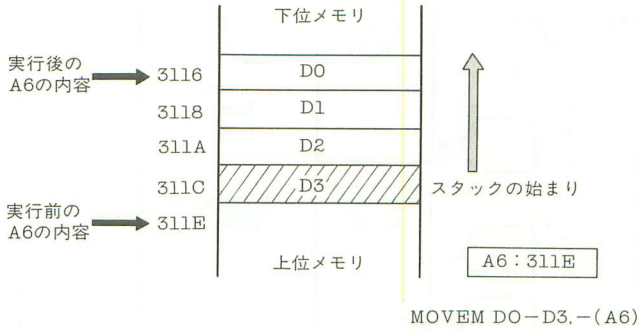



図 3・6 プリデクリメントを用いたプッシュ操作

3・4 キューの構成

キューとは，先入れ先出し（FIFO）で用いられる配列データ構造のメモリをいいます．MC68000では，ポストインクリメントまたはプリデクリメント・アドレス・レジスタ間接のアドレッシング・モードを利用することにより，ユーザのキューを作成し操作することができます．

キューは，データを先端から取り入れ，終端から取出しを行うために2個のアドレス・レジスタ（A0～A6のうち2個）を使用します．終端を指すアドレス・レジスタをプット・ポインタと呼び，先端を指すアドレス・レジスタをゲット・ポインタと呼びます．

また，プット・ポインタのアドレス更新とともにゲット・ポインタの指すアドレスも更新され，常にポインタ間のバッファ容量を一定に保ちます．

これはおかしくないか？これでは キューに新しいデータを置くと，まだ取得されていないデータが削除されることになってしまう。

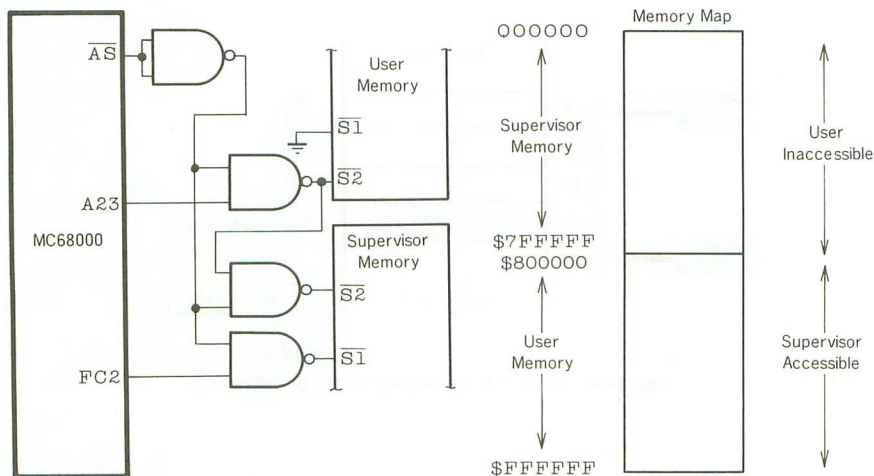



図 3・7 スーパバイザ・メモリ・アクセスにおけるユーザ保護

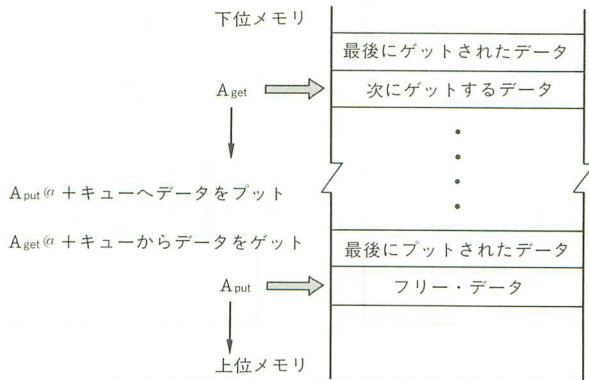



図 3・8 アドレスの上位へ向うキュー操作

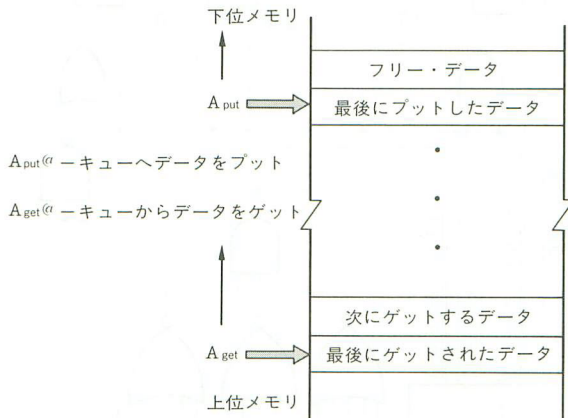



図 3・9 アドレスの下位へ向かうキュー操作

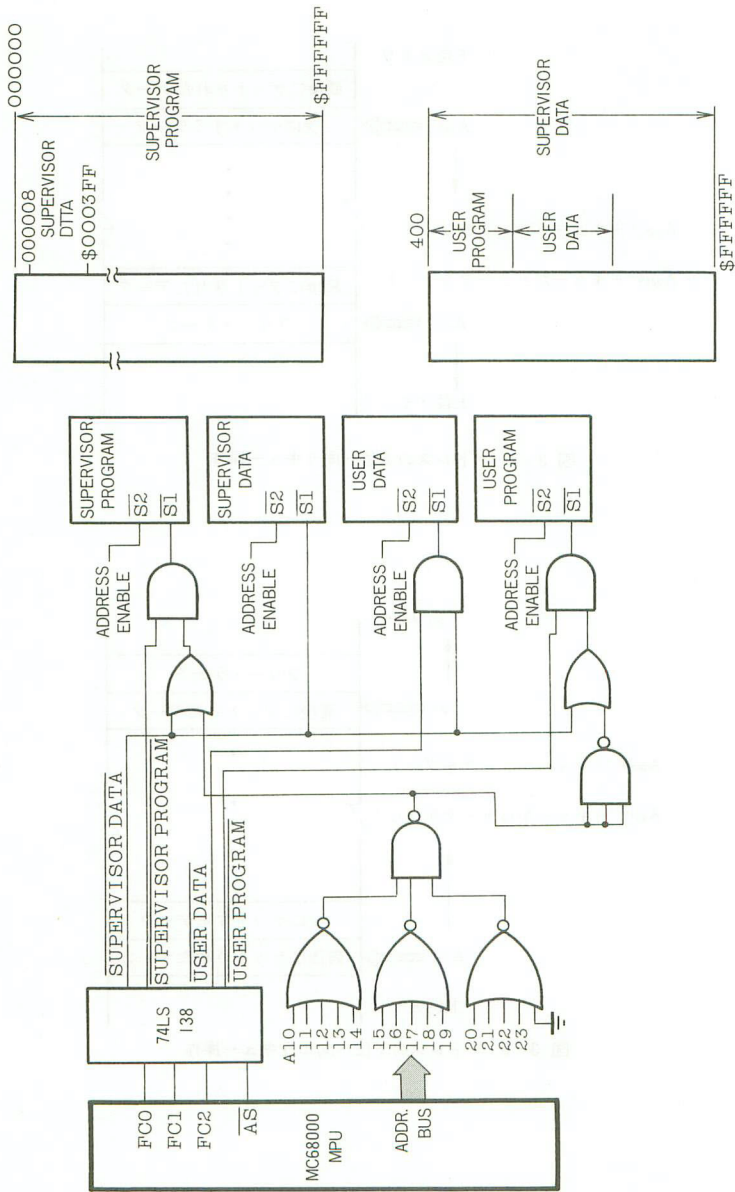



図 3・10 デュアル・メモリ・マップ

4. 入/出力の構成

68000 全体の入出力信号を図示し，入出力に関するシグナル名と，その機能について説明しています．データ・バスとデータ・ストローブの関係や，何種類かのバス・サイクルとシグナルの関係，およびデータ転送の関係も図示しています．

4・1 バス・シグナル

入/出力におけるバス・シグナルは，機能別にアドレス・バスとデータ・バスの二つの機能に分けられます．アドレス・バスとデータ・バスは，それぞれ独立した並列バスであり，非同期動作とMC6800系の同期式でデータを転送（送/受信）します．またどのようなバス・サイクルでも，バス・マスタは，すべての信号動作をそのバス・サイクル内で完了させなくてはなりません．

68000MPUと外部デバイス間のデータは，16本のデータ・バス（D0～D15）を通して行われます．したがって，メモリや入/出力間でやりとりできるデータは最大16ビットです．また，16ビットより大きいデータを転送するためには，2回以上の転送が必要となります．

〔1〕 **データ・バス**

- D0からD15のデータ・バスは，16ビットの双方向性3ステート・バスです．
- バイト長やワード長のデータ転送（送/受信）を行います．
- 割込みアクノリッジ・サイクル中は，外部チップからデータ・ラインD0～D7にベクタ番号が入ります．

〔2〕 **アドレス・バス**

- A1からA23のアドレス・バスは，23ビットの単方向性3ステート・バスです．
- 16Mバイト（16 777 216バイト）または8Mワード（8 388 600ワード）のデータを直接アドレス可能です．
- アドレス・ストローブライン（$\overline{AS}$）をアサートすることによりアドレス・バスに意味のあるアドレスを示します．
- 割込みサイクルを除いたすべてのサイクルにおいて，バス・オペレーション用のアドレスを決定します（6・1節の割込み参照）．

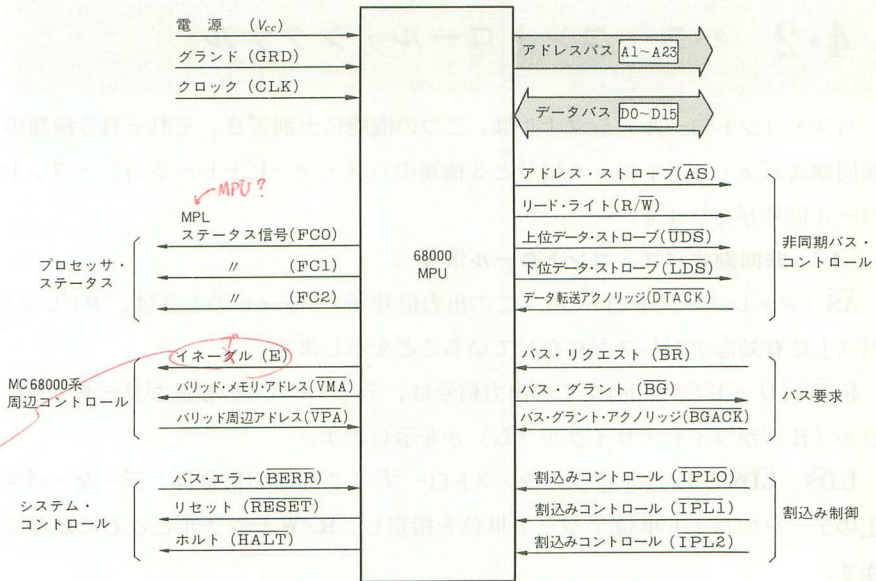



図 4・1　MC68000の入/出力信号

表 4・1　データ・バスのデータ・ストローブ・コントロール

$\overline{UDS}$	$\overline{LDS}$	R/$\overline{W}$	D8~D15	D0~D7
H	H	-	(意味なしデータ)	(意味なしデータ)
L	L	H	データ・ビット8~15	データ・ビット0~7
H	L	H	(意味なしデータ)	データ・ビット0~7
L	H	H	データ・ビット8~15	(意味なしデータ)
L	L	L	データ・ビット8~15	データ・ビット0~7
H	L	L	(データ・ビット0~7)	データ・ビット0~7
L	M	L	データ・ビット8~15	(データ・ビット8~15)

4・2 バス・コントロール・シグナル

バス・コントロール・シグナルは，二つの機能に大別でき，それぞれ5種類の非同期式バス・コントロール信号と3種類のバス・アービトレーション・コントロール信号があります．

〔1〕 非同期式バス・コントロール信号

$\overline{\mathbf{AS}}$（アドレス・ストローブ）：この出力信号がアクティブのときは，アドレス・バス上に有効なアドレスが存在していることを示します．

R/$\overline{\mathbf{W}}$((リード/ライト)：この出力信号は，データ・バスの転送がリード・サイクル（H）かライト・サイクル（L）かを示します．

$\overline{\mathbf{UDS}}$，$\overline{\mathbf{LDS}}$（上位/下位データ・ストローブ）：この出力信号は，データ・バス上のデータのバイト単位やワード単位を指定し，R／$\overline{\mathrm{W}}$ シグナルとともに動作します．

$\overline{\mathbf{DTACK}}$（データ転送アクノリッジ）：この入力信号は，データ転送が完了していることを示します．また，MPU がリード・サイクルやライト・サイクル中に $\overline{\mathrm{DTAK}}$ を認知すると，データをラッチしてバス・サイクルを終了させます．

〔2〕 バス・アービトレーション・コントロール信号

$\overline{\mathbf{BR}}$（バス・リクエスト）：この入力信号は，他のデバイスがバス・マスタになることを MPU に要求していることを示し，すべてのバス・マスタとなりうるデバイスをワイヤード OR にします．

$\overline{\mathbf{BG}}$（バス・グラント）：この出力信号は，MPU のバス・サイクルの終了した時点でバス制御を解除し，他のデバイスのバス・マスタの要求を移すことを示します．

* A0：68000内部シグナルでアドレス・バス・ビット"0"に相当し，A0がセットのときは下位データ・ストローブ(LDS)がセットし，データ・バスに"D0～D7"が乗る.

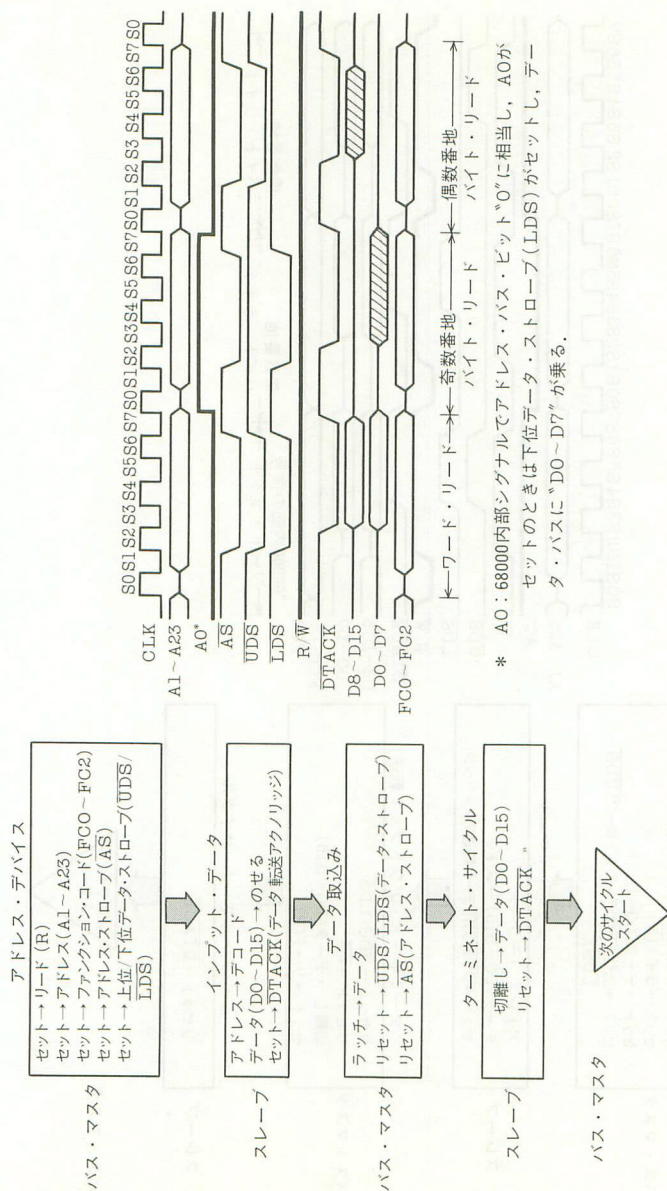



図 4・2 ワード・リード・サイクル・フローとタイミング

図 4・3 ワード・ライト・サイクル・フローとタイミング

リクエスト・バス	**リクエスト・バス** セット→バス・リクエスト($\overline{BR}$)
プロセッサ	**グラント・バス・アービトレーション(調停)** リセット→バス・グラント($\overline{BG}$)
リクエスト・バス	**アクノリッジ・バス・マスタ権** 1) 次のバス・マスタによって外部アービトレーションが決まる． 2) 進行中のサイクルが終了するまで次のバス・マスタは待つ． 3) 次のバス・マスタは$\overline{BGACK}$(バス・グラント・アクノリッジ)によりセットされる． 4) リセット→バス・リクエスト($\overline{BR}$)
プロセッサ	**ターミネート・アービトレーション** リセット→バス・グラント($\overline{BG}$)
リクエスト・バス	**バス・マスタ・オペレート** データ転送(リード・ライト・サイクル)→実行
リクエスト・バス	**バス・マスタ権　解除** リセット→$\overline{BGACK}$(バス・グラント・アクノリッジ)
プロセッサ	〔再アービトレーションかプロセッサ・オペレーションへ〕

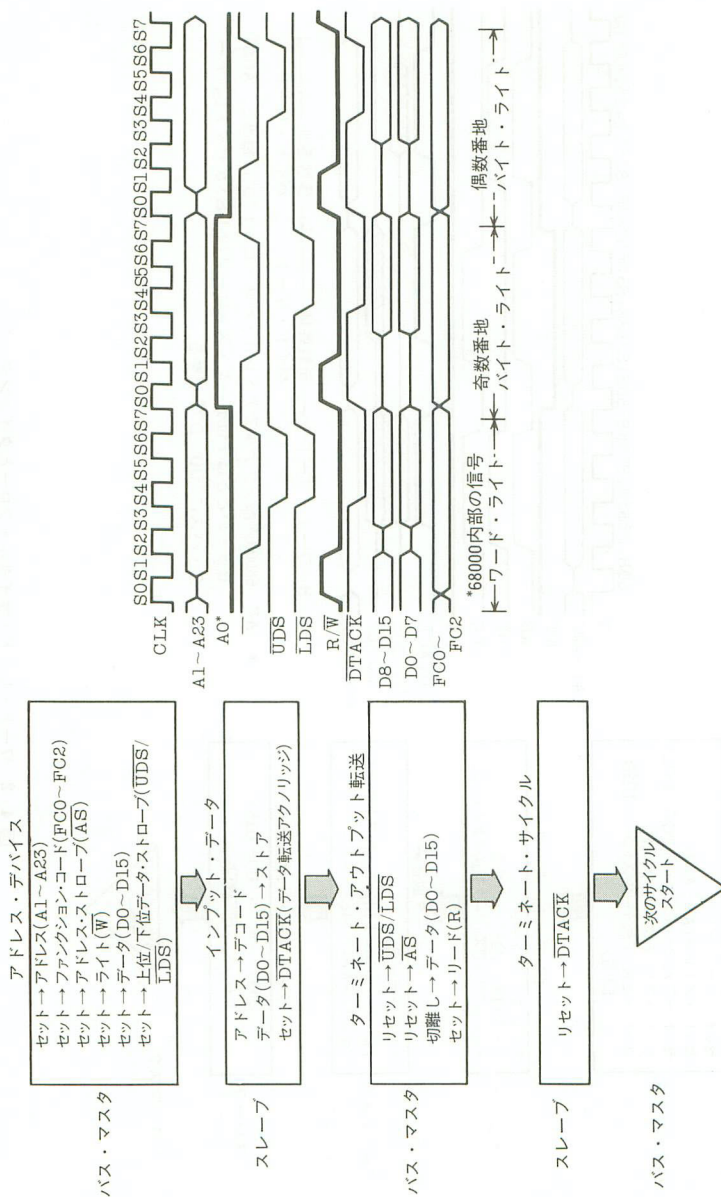
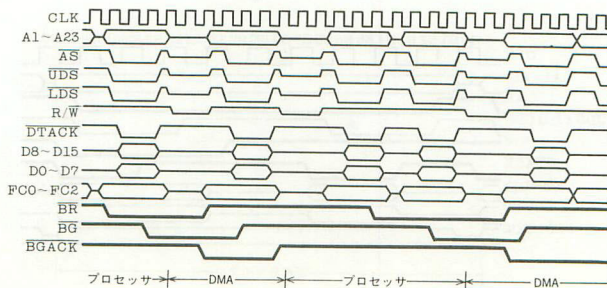
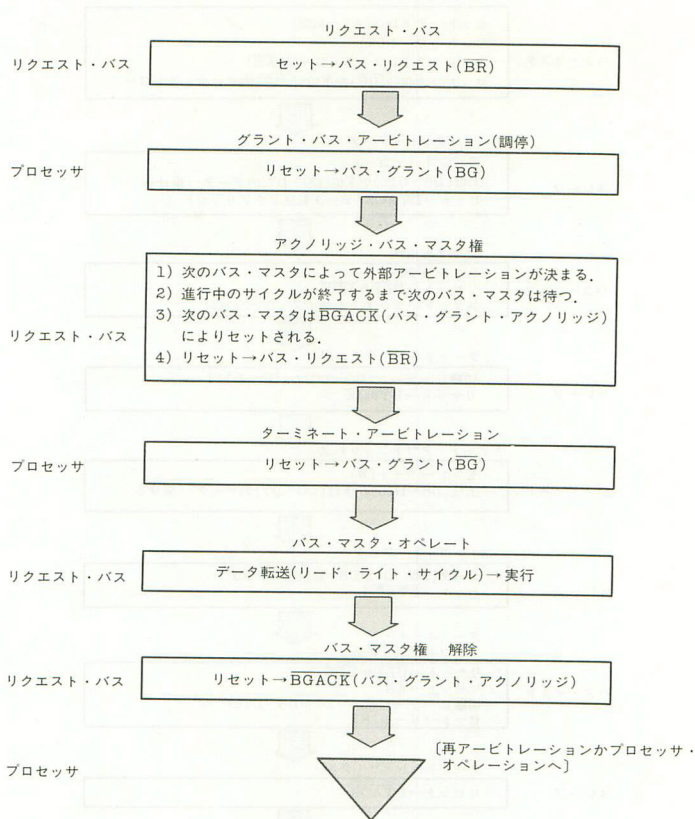



図 4・4　バス・アービトレーション・サイクル・フローとタイミング

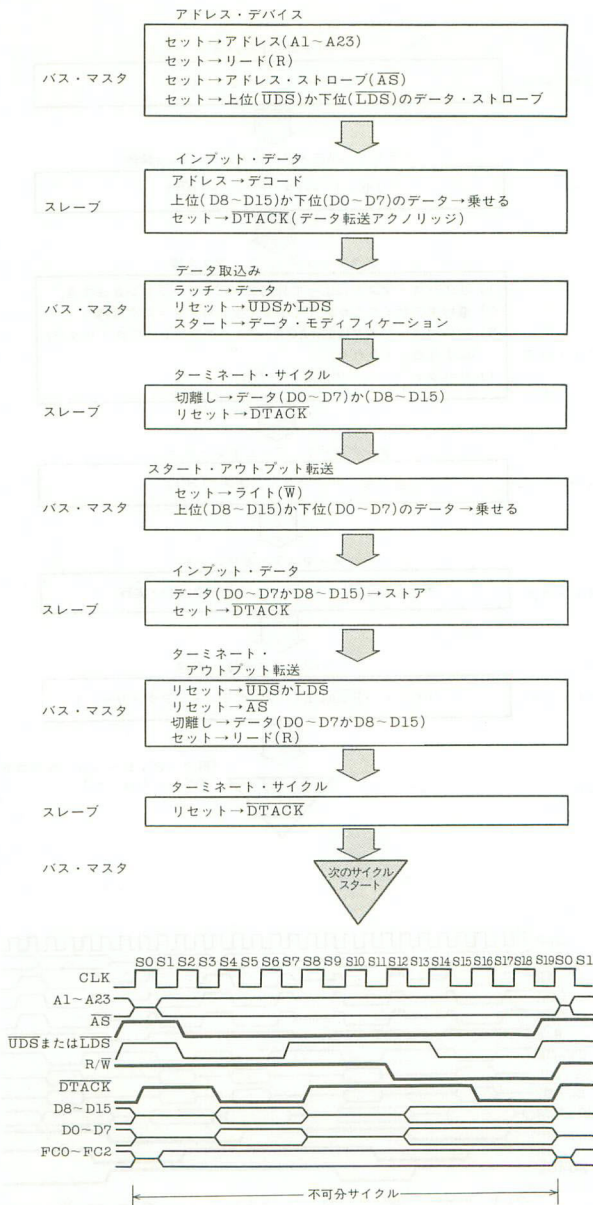



図 4・5 リード・モディファイ・ライト・サイクル・フローとタイミング

〔3〕 バス・アービトレション

バス・アービトレションは，プロセッサと外部デバイス間でのやりとりに，バス要求（$\overline{BR}$），バス・グラント（$\overline{BG}$），バス応答（$\overline{BGACK}$）の三つのシグナルにて制御されています．

バス・アービトレション・ユニットの状態図を図4·6に示します．

a．バス要求（$\overline{BR}$）　バス・マスタになりうる外部デバイスは，バス要求信号（$\overline{BR}$）をアサートすることによって，プロセッサに外部デバイスが外部バスのコントロールを要求していることを知らせます．

b．バス・グランド（$\overline{BG}$）　バス要求がアサートされることにより，プロセッサは，可能な限り速やかにバス・グラント（$\overline{BG}$）をアサートし，バス・アービトレション承認をします．

c．バス・マスタの応答（$\overline{BGACK}$）　バス・グラントがアサートされると要求を出した外部デバイスは，バスマスタ応答（$\overline{BGACK}$）をアサートし新しいバスマスタとなります．

プロセッサ・バス・サイクルにおけるバス・インアクティブどきの特殊例としておのおののバス・アービトレションについて説明します．

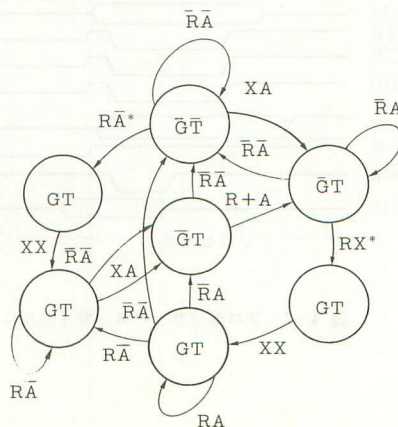



図 4·6　MC68000バス・アービトレーション・ユニット図

図4·7は，プロセッサのバス・サイクル内でのバス・アービトレーションのタイミング図で，バス要求デバイス信号をアサートし3ステートからのバスの解放とプロセッサの次のバス・サイクルまでを示します．

図4·8は，バスがインアクティブなときのバス・アービトレーションのタイミング図で，乗算や割算命令などの内部動作実行時のシーケンスを示します．

図4·9は，プロセッサのバス・サイクルの特別な場合のバス・アービトレーションのタイミング図で，MPUがバス・サイクルを開始したが，まだアドレス・ストローブ信号($\overline{AS}$)がアサートされていないときにバス要求が起きてもバスグラントは次の立ち上がりではアサートされず，内部でのアサートから二つ目の立ち上がりでバスグラントがアサートされます．

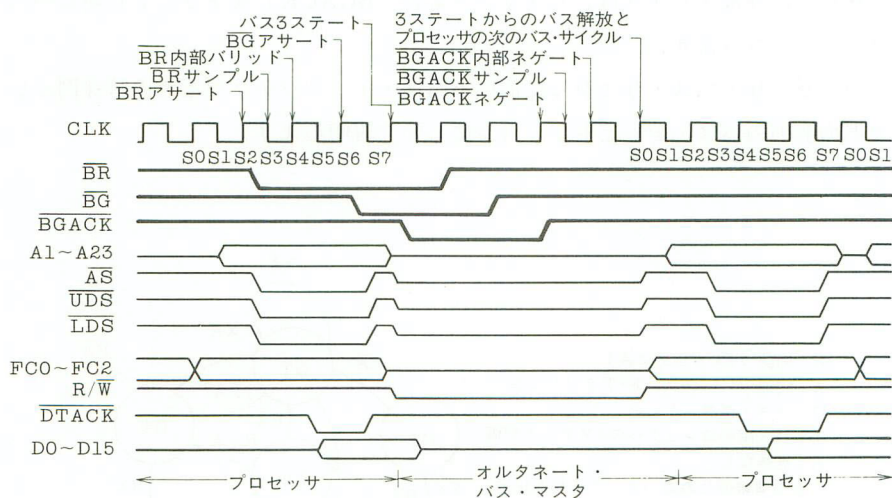



図4·7 プロセッサ・バス・サイクルにおけるバス・アービトレーション

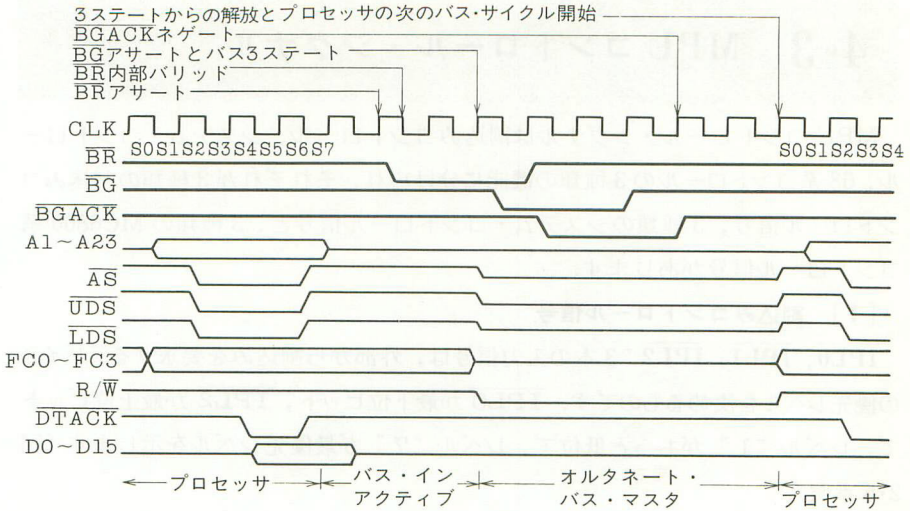



図 4・8　バス・インアクティブにおけるバス・アービトレーション

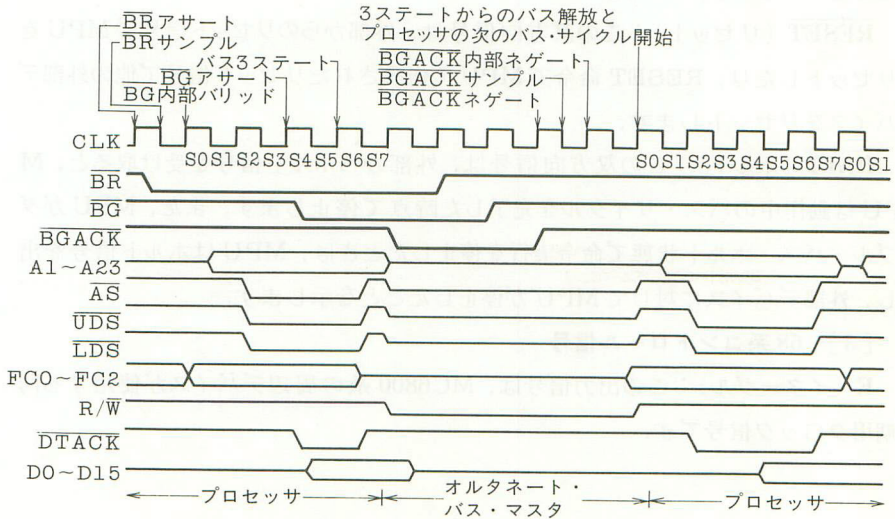



図 4・9　特殊例のプロセッサ・バス・サイクルにおけるバス・アービトレーション

4・3 MPUコントロール・シグナル

MPUコントロール・シグナルは割込みコントロール，システム・コントロール，68系コントロールの3種類の機能に分けられ，それぞれが3種類の割込みコントロール信号，3種類のシステム・コントロール信号と，3種類のMC6800系コントロール信号があります．

［1］ 割込みコントロール信号

$\overline{\mathrm{IPL0}}$，$\overline{\mathrm{IPL1}}$，$\overline{\mathrm{IPL2}}$：3本の入力信号は，外部から割込みを要求するデバイスの優先レベルを決めるものです．$\overline{\mathrm{IPL0}}$が最下位ビット，$\overline{\mathrm{IPL2}}$が最上位ビットで，レベル“1”がもっと低位で，レベル“7”が最優先レベルを示します（図2・5参照）．

［2］ システム・コントロール信号

$\overline{\mathrm{BEERR}}$（バス・エラー）：この入力信号は，実行中のバス・サイクルに問題が生じたことをMPUに知らせます．

$\overline{\mathrm{RESET}}$（リセット）：この双方向信号は，外部からのリセット信号でMPUをリセットしたり，RESET命令でMPUで作成されたリセット信号で他の外部デバイスをリセットします．

$\overline{\mathrm{HALT}}$（ホルト）：この双方向信号は，外部からホルト信号を受け取ると，MPUは動作中のバス・サイクルを完了した時点で停止します．また，MPUがダブル・バス・ホルト状態で命令実行を停止したときは，MPUはホルト信号を出し，外部デバイスに対してMPUが停止したことを示します．

［3］ 68系コントロール信号

E（イネーブル）：この出力信号は，MC6800系の周辺デバイスが使用する同期用クロック信号です．

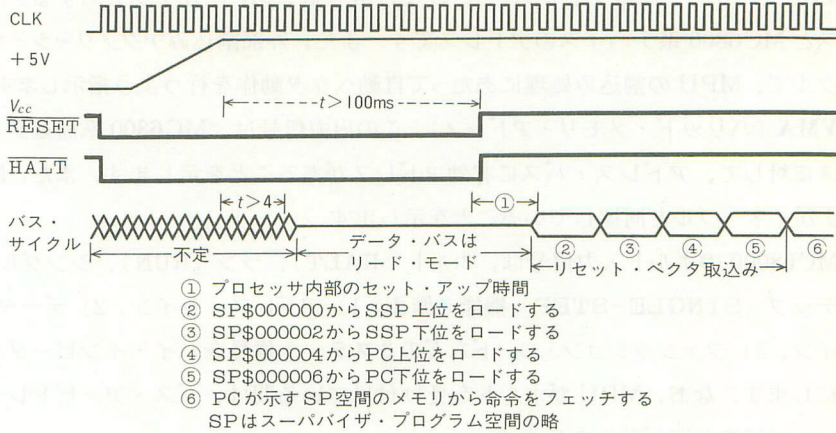



図 4・10 リセット・オペレーションのタイミング

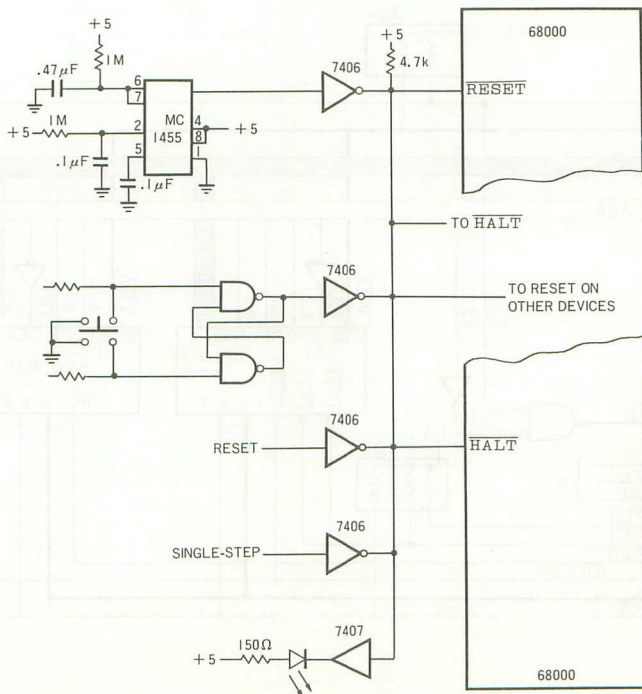



図 4・11 リセットとホルト回路

$\overline{\text{VPA}}$（バリッド周辺アドレス）：この入力信号は，MPU がアクセスするデバイスと MC6800 系デバイスのアドレスです．また，外部割込みアクノリッジ・サイクルで，MPU の割込み処理にあたって自動ベクタ動作を行うよう指示します．

$\overline{\text{VMA}}$（バリッド・メモリ・アドレス）：この出力信号は，MC6800 系周辺デバイスに対して，アドレス・バスに有効アドレスがあることを示します．また，MPU がイネーブルと同期していることを示します．

MC68000 のホルト入力信号は，ホルト（HALT），ラン（RUN），シングル・ステップ（SINGLE-STEP）機能を備え，1）アドレス・ライン，2）データ・ライン，3）ファンクション・コードなどの**3 ステート信号**をハイ・インピーダンスにします．なお，MPU がホルトを受け付けている間は，バス・アービトレーションは通常と同じ動作をします．

ホルト機能とハードウェアによる機能を用いることにより，ハードウェア・デ

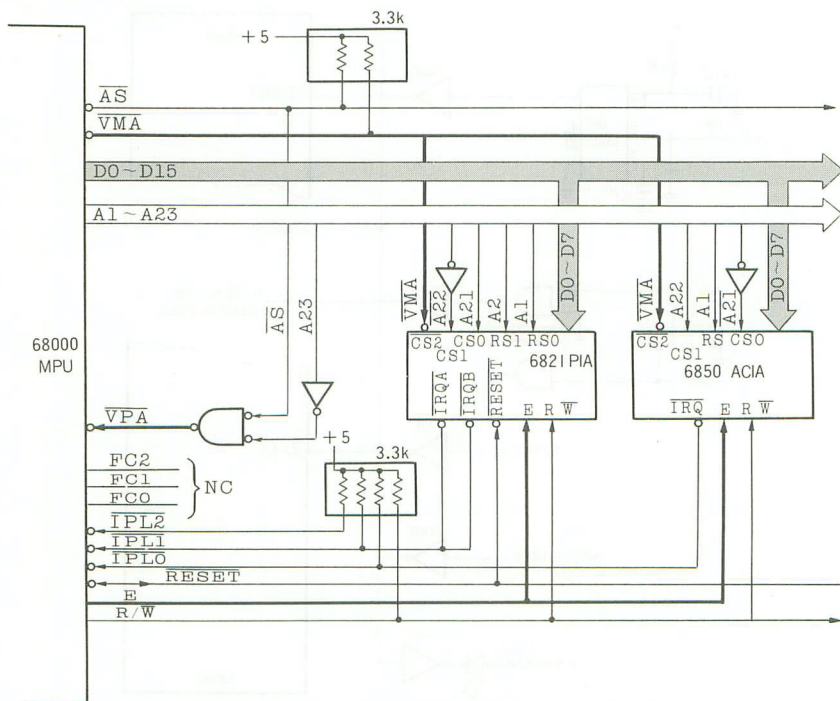



図 4・12　MC6800 系のインタフェース例

バッガ（シングル・ステップ・モード）でプロセッサを１バス・サイクルまたは１命令ずつトレースすることにより，これらの機能やソフトウェア・デバッグ機能などフレキシビリティのある高いデバッグ能力を備えています．

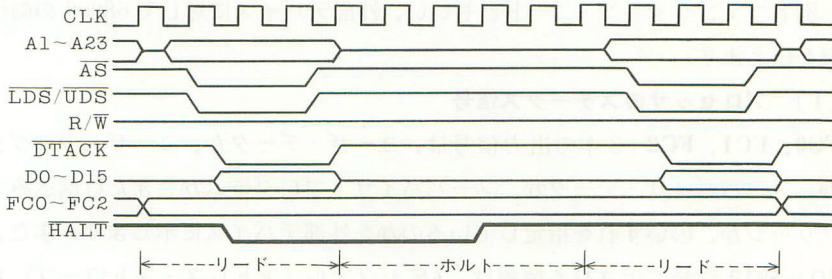



図 4・13　バス・エラーでないときのホルト・タイミング

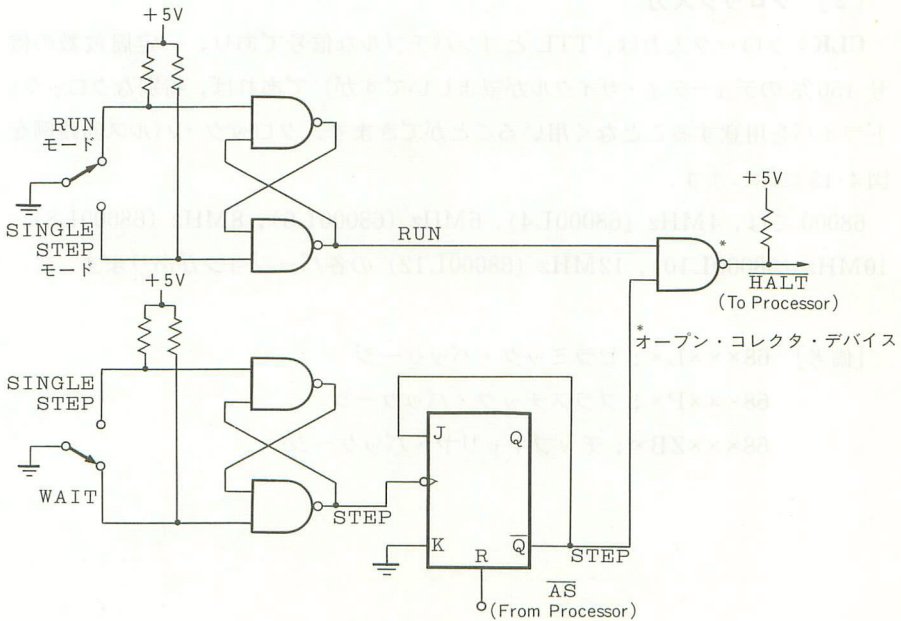



図 4・14　シングル・ステップ回路

4・4 MPU ステータス・シグナルとクロック

プロセッサのメモリ・アクセス状態と割込み受付けを示す3種のステータス信号，別名ファンクション・コードともいい，外部デバイスに対して68000の動作状況を伝えます．

［1］ **プロセッサのステータス信号**

FC0，FC1，FC2：3本の出力信号は，ユーザ・データか，ユーザ・プログラムか，スーパバイザ・データか，スーパバイザ・プログラムか，または割込みアクノリッジか，のいずれを指定しているのかを外部デバイスに示します．また，FC0～FC2信号で示される情報は，$\overline{\mathrm{AS}}$シグナル（アドレス・ストローブ）がアクティブであれば常に有効です（表4・2参照）．

［2］ **クロック入力**

CLK：クロック入力は，TTLとコンパチブルな信号であり，一定周波数の信号（50％のデューティ・サイクルが望ましいですが）であれば，特別なクロック・ドライバを用意することなく用いることができます．クロック・パルス回路例を図4・15に示します．

68000では，4MHz（68000L4），6MHz（68000L6），8MHz（68000L8），10MHz（68000L10），12MHz（68000L12）の各バージョンがあります．

［備考］ 68×××L×；セラミック・パッケージ
68×××P×；プラスチック・パッケージ
68×××ZB×；チップキャリヤ・パッケージ

表 4・2 ファンクション・コードと機能の関係

FC2	FC1	FC0	サイクル・タイプ（機能）
0	0	0	—
0	0	1	ユーザ・データ
0	1	0	ユーザ・プログラム
0	1	1	—
1	0	0	—
1	0	1	スーパバイザ・データ
1	1	0	スーパバイザ・プログラム
1	1	1	割込みアクノリッジ

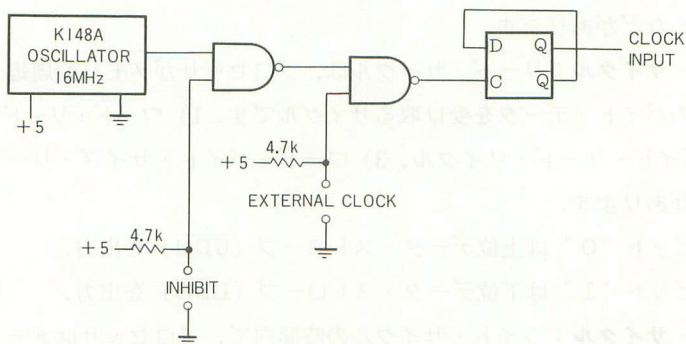



図 4・15 クロック・パルス供給回路

4・5 データ転送

8 ビット MPU6800 は，同期式のデバイスとしかデータの送受信ができませんでしたが，16 ビット MPU68000 では，同期式でも非同期式でも両方式のデバイスとインタフェースが可能です.

つまり，アクセス・タイムの遅いものには，それなりにゆっくりと，アクセス・タイムの速いものには，それなりに速く合わせるというように，実行速度を順応させることができます.

68000 は，データ転送において通常のリード・サイクルとライト・サイクル，マルチ・プロセッサ構成時のリード・モディファイ・ライト・サイクル，6800 周辺サイクルなどがあります.

リード・サイクル：リード・サイクルは，プロセッサがメモリや周辺デバイスから複数のバイト・データを受け取るサイクルです. 1）ワード・リード・サイクル，2）バイト・リード・サイクル，3）ワード・バイト・サイズ・リード・サイクルなどがあります.

A0 ビット“0”は上位データ・ストローブ（$\overline{\text{UDS}}$）を出力.

A0 ビット“1”は下位データ・ストローブ（$\overline{\text{LDS}}$）を出力.

ライト・サイクル：ライト・サイクルの時間内で，プロセッサはメモリや周辺デバイスに複数のバイト・データを送り出すサイクルです. 1）ワード・ライト・サイクル，2）バイト・ライト・サイクル，3）リード・モディファイ・ライトサイクルなどがあります.

A0 ビット“0”は上位データ・ストローブ（$\overline{\text{UDS}}$）を出力.

A0 ビット“1”は下位データ・ストローブ（$\overline{\text{LDS}}$）を出力.

〔1〕 **非同期式バス・コントロール**　データ転送のコントロールは，2 本の制御シグナルである $\overline{\text{UDS}}$ と $\overline{\text{LDS}}$ によって，バイト単位やワード単位のデータを指定します.

通常のリード・サイクルは **4 クロック**・サイクルが必要であり，ライト・サイクルのときは **5 クロック**・サイクルが必要です.

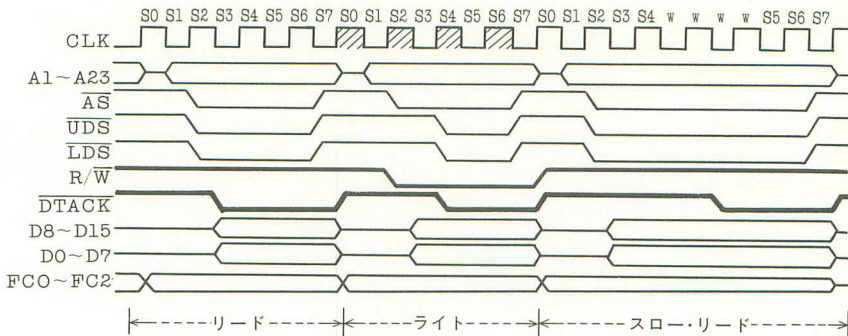



図 4・16 リード・サイクルとライト・サイクルのタイミング

〔2〕 **同期式バス・コントロール** MC6800系のバスサイクルは，6800系周辺デバイス（PIAやACIA）が使用できるように，同期式バス・サイクルを非同期式の68000バス・サイクルに変更させます．インタフェースするための同期式バス制御ラインは，**E**と非同期式バス信号$\overline{\text{DTACK}}$に代わる信号として$\overline{\textbf{VPA}}$，そして$\overline{\textbf{VMA}}$の3種類のシグナルです（図2・5，図4・12参照）．

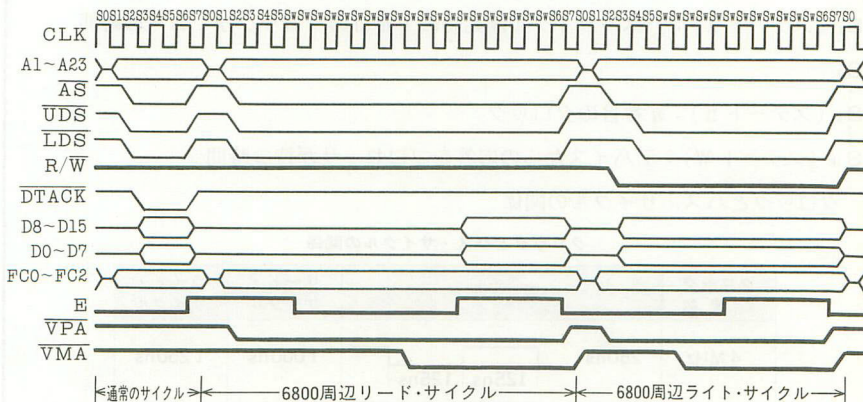



図 4・17 6800系周辺バス・サイクルのタイミング

クロックの見方

S_0（ステート0）：バス・サイクル開始時の1番目のクロックが“H”の時間

S_1（ステート1）：バス・サイクル開始時の1番目のクロックが“L”の時間

S_2

⋮

S_n（ステートn）：n 番目のクロック

⋮

S_W（ステートW）：デバイスからの応答をプロセッサが待つ時間

クロックとバス・サイクルの関係

クロックとバス・サイクルの関係

クロック周波数	周　期	リード・サイクル	ライト・サイクル
4 MHz	250ns（125ns / 125ns）	1 000ns	1 250ns
8 MHz	125ns（62.5ns / 62.5ns）	500ns	625ns
10 MHz	100ns（50ns / 50ns）	400ns	500ns

5. アドレッシング・モード

68000の特長であるアドレッシング・モードは，モードを多数備え，アドレス計算や転送命令の省略化による効率良いプログラミングの作成と速度の向上を提供します．

7種類のアドレス・モードを各モード単位で具体例にて図示し，配列データ処理法，プログラムの再配置法，レジスタの自動的な増減法，メモリ・スタックの作成法などを解説しています．

5・1 実効アドレス

各命令のオペランドのバリエーションを決定するのに必要な要素として，アドレッシング・モードがあげられます．68000 のアドレッシング・モードは六つの基本グループに分けられ，総計 14 種類のアドレッシング・モードを備え，その種類の豊富さと柔軟さによって，プログラミングのステップ数の省略や効率の向上，実効速度の向上ができます．また，繁雑になりがちなものも整然とした管理につながっていきましょう．

14 種類のアドレッシング・モードは，2 種のレジスタ直接，5 種類のレジスタ間接，2 種類のアブソリュート，2 種類のプログラム・カウンタ相対，2 種類のイミディエート，1 種類のインプライドを備えています（表 5·1 参照）．

ほとんどの命令では，オペランドの位置はオペレーション・ワードの実行アドレス・フィールドを使用します．実行アドレス・フィールドは，レジスタ・フィールド（ビット 0～2）とモード・フィールド（ビット 3～5）からなっています．レジスタ・フィールドはレジスタ番号を指定し，モード・フィールドは，いろいろなモードを選択します（表 5·2 参照）．

15					10					5	2	0
x	x	x	x	x	x	x	x	x	x	実効アドレス	
										モード	レジスタ

図 5・1　単一実効アドレス命令のオペレーション・ワードの一般的な型

表 5・1　基本グループとアドレッシング・モード

基本型	モード	機能
レジスタ直接	1）データ・レジスタ直接 2）アドレス・レジスタ直接	$EA=D_n$ $EA=A_n$
レジスタ間接	3）アドレス・レジスタ間接 4）ポストインクレメント 5）プリデクレメント 6）オフセット付き 7）インデックスおよびオフセット付き	$EA=(A_n)$ $EA=(A_n), A_n \leftarrow A_n+N$ $A_n \leftarrow A_n-N, EA=(A_n)$ $EA=(A_n)+d_{16}$ $EA=(A_n)+(X_n)+d_8$
アブソリュート（絶対）	8）アブソリュート・ショート 9）アブソリュート・ロング	EA=（次のワード） EA=（次の２ワード）
プログラム・カウンタ相対	10）オフセット付き 11）インデックスおよびオフセット付き	$EA=(PC)+d_{16}$ $EA=(PC)+(X_n)+d_8$
イミディエート	12）イミディエート 13）クイック・イミディエート	DATA=次のロード Inherent DATA
インプライド（合意）	14）インプライド・レジスタ	EA=SR, USP, SP, PC

EA＝実効アドレス
A_n＝アドレス・レジスタ
D_n＝データ・レジスタ
SR＝ステータス・レジスタ
PC＝プログラム・カウンタ

X_n＝インデックス・レジスタとして用いられるアドレス・レジスタまたはデータ・レジスタ
（ ）＝内容
←＝転送

d_8 ＝8ビット・オフセット
d_{16} ＝16ビット・オフセット
N＝1（バイトのとき）
　2（ワードのとき）
　4（ロング・ワードのとき）

表 5・2　アドレッシング・モードのコード別コード表

アドレス・モード	モード	レジスタ
データ・レジスタ直接	000	レジスタ番号
アドレス・レジスタ直接	001	レジスタ番号
アドレス・レジスタ間接	010	レジスタ番号
ポストインクレメント・アドレス・レジスタ間接	011	レジスタ番号
プリデクリメント・アドレス・レジスタ間接	100	レジスタ番号
ディスプレースメント・アドレス・レジスタ直接	101	レジスタ番号
インデックス・アドレス・レジスタ間接	110	レジスタ番号
アブソリュート（絶対）・ショート	111	000
アブソリュート（絶対）・ロング	111	001
ディスプレースメント・プログラム・カウンタ	111	010
インデックス・プログラム・カウンタ	111	011
イミディエート/ステータス・レジスタ	111	100

5・2 レジスタ直接モード

レジスタ直接モードには，データ・レジスタ直接とアドレス・レジスタ直接の2種類があります．実効アドレッシングは，8個のデータ・レジスタ（D0～D7）または8個のアドレス・レジスタ（A0～A7）の中のいずれか1個をオペランドとして指定できます．

［1］ **データ・レジスタ直接**　　オペランド・データは，実効アドレス・レジスタ・フィールドで指定したデータ・レジスタに格納されます．

機　能：EA=Dn
PTL 表記：Dn
モード：000
レジスタ：n
データ・レジスタDn：オペランド

例　MOVE　D0，$1000

［2］ **アドレス・レジスタ直接**　　オペランド・データは，実効アドレス・レジスタ・フィールドで指定したアドレス・レジスタに格納されます．

機　能：EA=An
PTL 表記：An
モード：001
レジスタ：n
アドレス・レジスタAn：オペランド

例　MOVE　A4，$202000

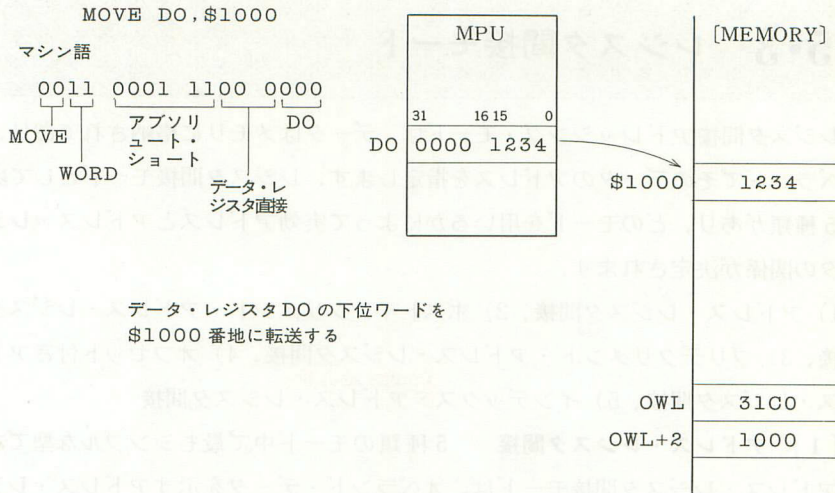



図 5・2 データ・レジスタ直接アドレッシング

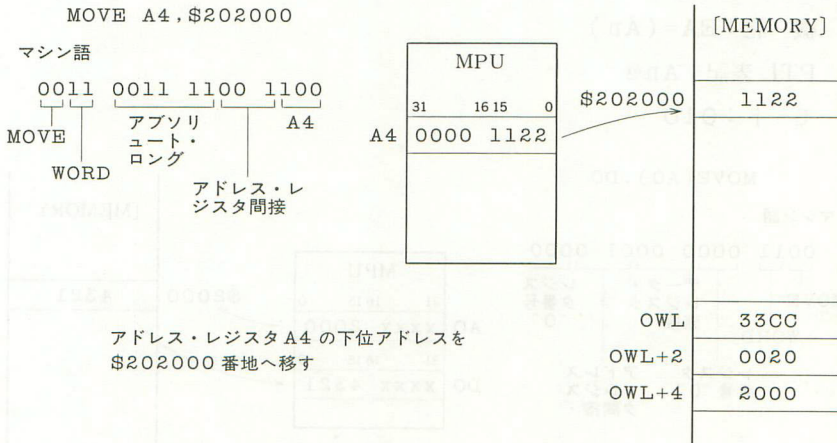



図 5・3 アドレス・レジスタ直接アドレッシング

5・3 レジスタ間接モード

レジスタ間接アドレッシング・モードは，データはメモリに格納されており，オペランドでそのデータのアドレスを指定します．レジスタ間接モードとして次の5種類があり，どのモードを用いるかによって実効アドレスとアドレス・レジスタの関係が決定されます．

1）アドレス・レジスタ間接，2）ポストインクリメント・アドレス・レジスタ間接，3）プリデクリメント・アドレス・レジスタ間接，4）オフセット付きアドレス・レジスタ間接，5）インデックス・アドレス・レジスタ間接

［1］ **アドレス・レジスタ間接**　5種類のモード中で最もシンプルな型であるアドレス・レジスタ間接モードは，オペランド・データを示すアドレス・レジスタ・フィールドで指定されるアドレス・レジスタに格納され，ジャンプ命令（JMP）やジャンプ・サブルーチン命令（JSR）を除いて，オペランドによってデータ位置を示します．

機　能：EA=(An)

PTL 表記：An@

モード：010

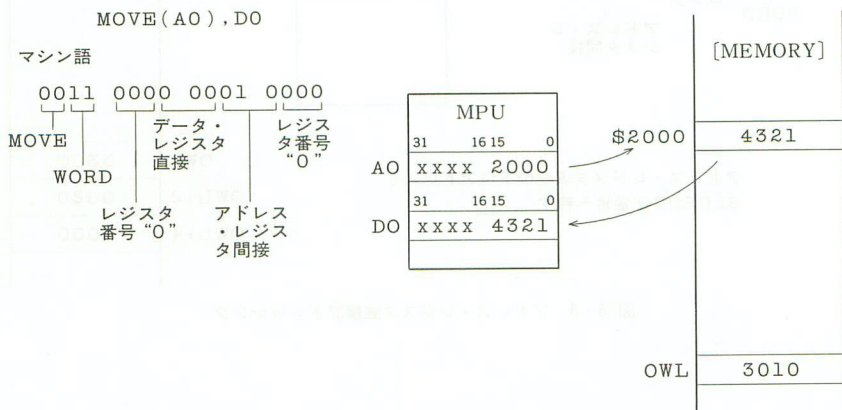



図 5・4　アドレス・レジスタ間接

```
PALIN:1    MC68000 ASM

  1                      *   DETECTING A 16-BIT PALINDROME
  2                      *
  3                      *A 16-BIT WORD POINTED TO BY A0
  4                      *IS CHECKED FOR SYMMETRY,I.E.,
  5                      *THE WORD READS THE SAME LEFT
  6                      *TO RIGHT AS IT READS RIGHT TO
  7                      *LEFT. IF IT IS DETERMINED TO BE
  8                      *A PALINDROME, THE BYTE TO WHICH
  9                      *THE STACK POINTER IS POINTING
 10                      *AFTER RTS WILL BE ALL 1'S
 11                      *
 12        00001000       ORG $1000
 13 001000 48E7E000      PALCHK MOYEM.L D0/D1/D2,-(SP)
 14 001004 7407           MOVEQ #7,D2        * 7 --> COUNTER
 15 001006 3010           MOVE (A0),D0       *TEST WORD --> D0.W
 16 001008 E248          AGAIN LSR 1,D0      *LSB,D0.W --> X,CCR
 17 00100A E351           ROXL 1,D1          *X,CCR --> LSB,D1.W
 18 00100C 51CAFFFA       DBRA D2,AGAIN      *DO THIS LOOP 8 TIMES
 19 001010 B200           CMP.B D0,D1        *IF MSB OF TST WRD=MIRROR IMAGE OF LSB
 20 001012 57EF0010       SEQ 16(SP)         *THN $FF --> 16(SP),ELS 00 --> 16(SP)
 21 001016 4CDF0007       MOVEM.L (SP)+,D0/D1/D2
 22 00101A 4E75           RTS
 23                       END

****** TOTAL ERRORS  0--  0

SYMBOL TABLE

AGAIN       001008 PALCHK      001000
```

PALINDROMチェック・プログラム例

レジスタ：n

アドレス・レジスタ An：［31 … 0］ メモリ・アドレス

↓

メモリ・アドレス：［オペランド］

例　MOVE(A0),D0

ポストインクリメントとプリデクリメントは，実行した後にアドレス・レジスタの値を増したり減らしたり自動的に更新し，メモリのデータを他の場所に，増加するアドレス順や減少するアドレス順に転送するのに有効です．

〔2〕 **ポストインクリメント**　オペランド・データを示すアドレスは，レジスタ・フィールドで指定されるアドレス・レジスタに格納されます．命令を実行した後に指定されたアドレス・レジスタでバイト，ワード，ロング・ワードのデータ・サイズの指定に応じて，指定されたアドレス・レジスタを1（バイト），2（ワード），4（ロング・ワード）を加算します．

機　能：An=An+N，EA=(An)

RTL 表記：An@+

モード：011

レジスタ：n

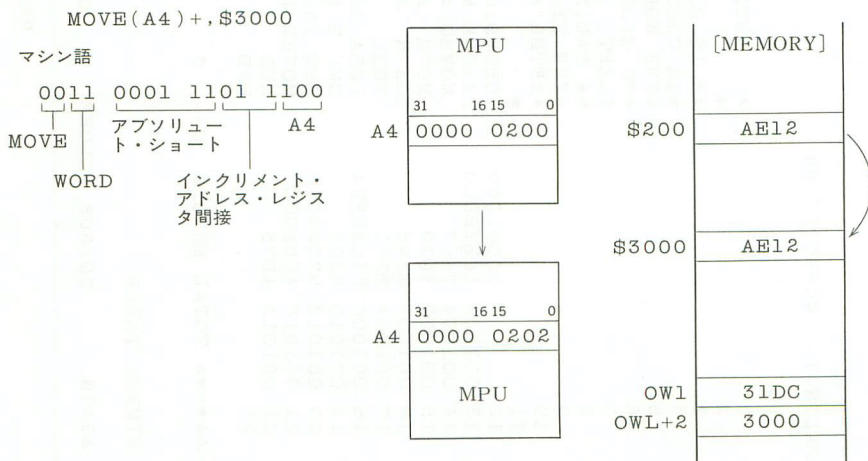



図 5・5　ポストインクリメント付きアドレス・レジスタ間接

アドレス・レジスタ

①オペランド長さ1，2，4

例 MOVE (A4)+，$3000

［3］ **プリデクリメント**　オペランドのアドレスは，レジスタ・フィールドで指定されるアドレス・レジスタに格納され，命令を実行する前に指定されたアドレス・レジスタで，バイト，ワード，ロング・ワードのデータ・サイズの指定に応じて，指定されたアドレス・レジスタを1（バイト），2（ワード），4（ロング・ワード）を減算します．

機　能：An=An－N，EA=(An)

RTL 表記：An@-

モード：100

レジスタ：n

例 MOVE －(A3)，$3000

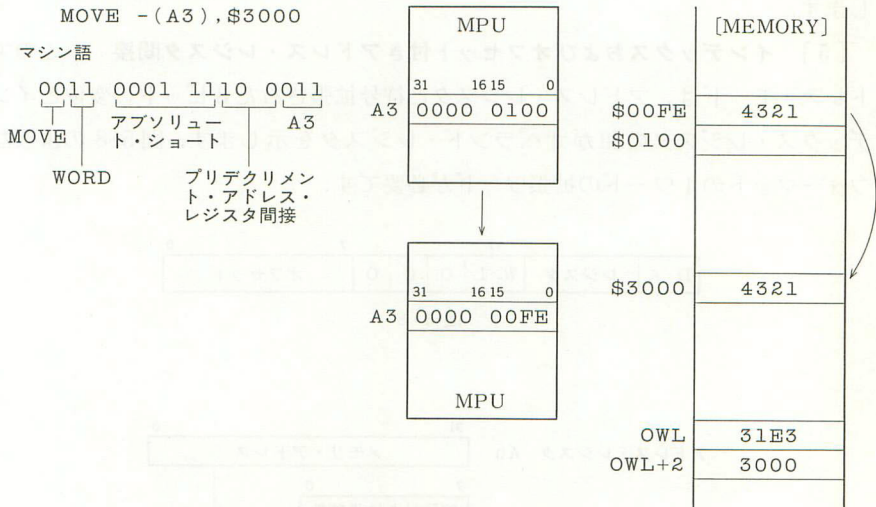



図 5·6　プリデクリメント付きアドレス・レジスタ間接

［4］ **オフセット付きアドレス・レジスタ間接**　このアドレス・モードは，アドレス・レジスタの内容と16ビット変位付き整数の和がオペランド・アドレスを示します．

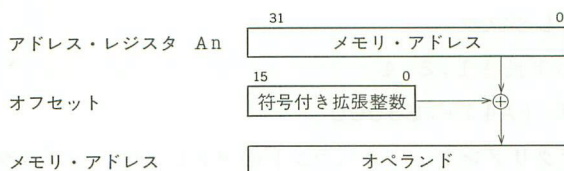



図 5・7

機　能：EA=(An)+d

RTL 表記：An@(d)

モード：101

レジスタ：n

アドレス・レジスタA0に先頭アドレス“2000”とすると，命令を実行することにより，先頭アドレスにオフセット“200”が加わり，実効アドレス“$2200”となって，アドレス内容“1234”をアドレス“$4000”番地に転送します．

〔5〕 **インデックスおよびオフセット付きアドレス・レジスタ間接**　このアドレス・モードは，アドレス・レジスタと符号拡張された8ビットの変位とインデックス・レジスタの和がオペランド・レジスタを示します．図5・8のようなフォーマットの1ワードの拡張ワードが必要です．

図 5・8

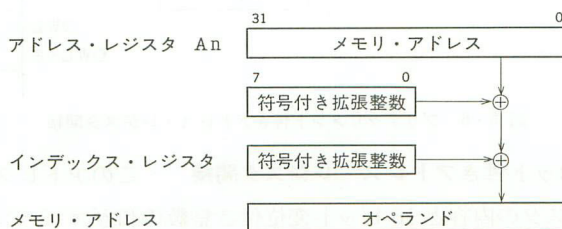



図 5・9

機　能：EA=(An)+(Ri)+d

RTL 表記：An@(d,Ri.W)

　　　　　An@(d,Ri.L)

インデックスは 16 or 32 ビットが選択できます.

モード：110

レジスタ：n

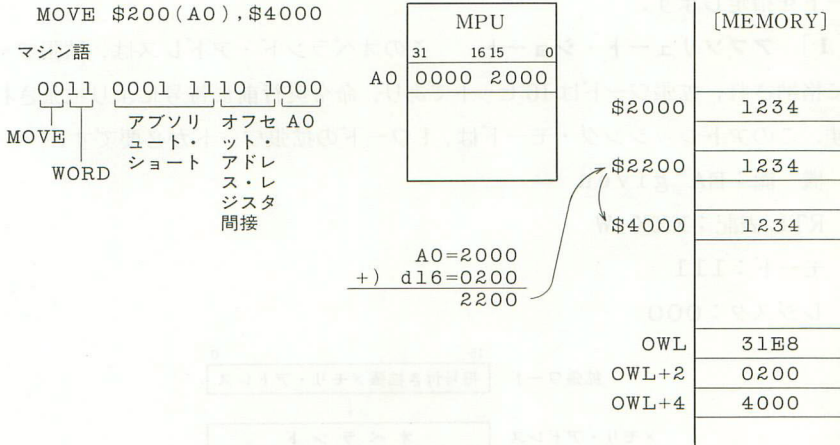



図 5・10　オフセット付きアドレス・レジスタ間接

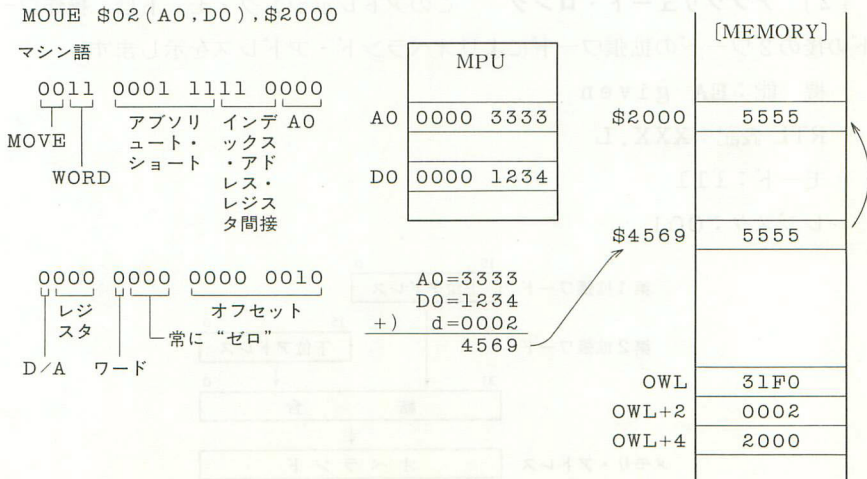



図 5・11　インデックスおよびオフセット付きアドレス・レジスタ間接

5・4　絶対（アブソリュート）モード

アブソリュート・アドレッシングは，ショートとロングの二つがあり，レジスタ番号の代わりに実効アドレス・フィールドを使用して，特定のアドレッシング・モードを指定します．

［1］　**アブソリュート・ショート**　　このオペランド・アドレスは，拡張ワードに格納され，拡張ワードは16ビットであり，命令実行前に符号により拡張されます．このアドレッシング・モードは，1ワードの拡張ワードが必要です．

機　能：EA given

RTL 表記：XXX.W

モード：111

レジスタ：000

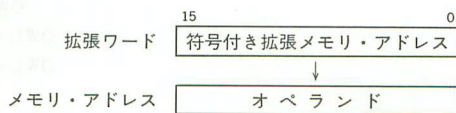



図 5・12　アブソリュート・ショート

［2］　**アブソリュート・ロング**　　このアドレッシング・モードは，操作ワードの後の2ワードの拡張ワードによりオペランド・アドレスを示します．

機　能：EA given

RTL 表記：XXX.L

モード：111

レジスタ：001

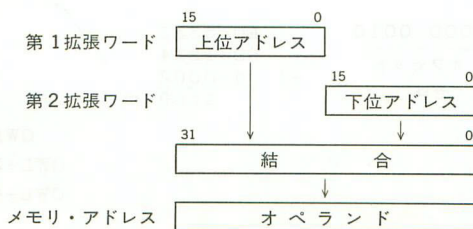



図 5・13　アブソリュート・ロング

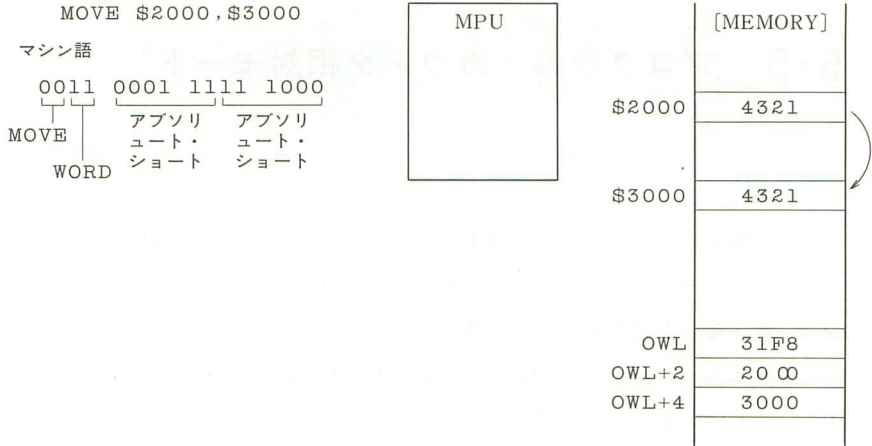



図 5・14 絶対（アブソリュート）ショート・アドレス

NEG $012000

マシン語

0100 0100 0111 1001

NEGインストラクション

サイズ

アブソリュート・ロング

MPU

[MEMORY]

$12000 0001→FFFF

OWL 4479

OWL+2 0001

OWL+4 2000

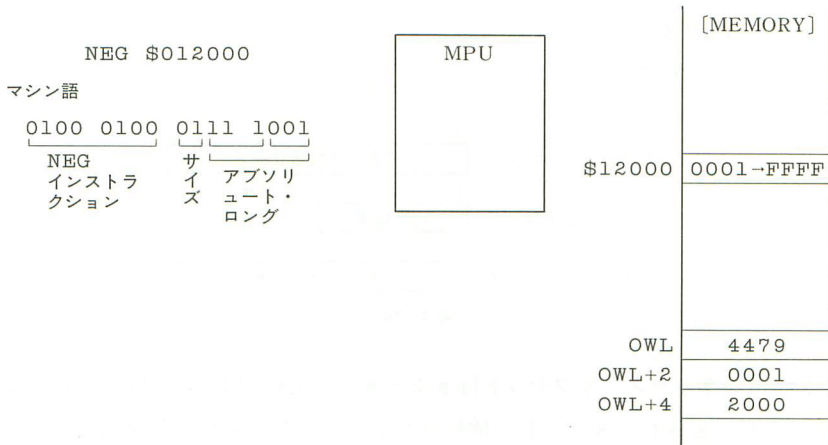



図 5・15 絶対（アブソリュート）ロング・アドレス

5•5 プログラム・カウンタ相対モード

プログラム・カウンタ相対モードは，プログラム・カウンタからの変位がオフセットにより決まるものと，オフセットとインデックスの両方によって決まるものとの2種類のモードがあり，いずれもプログラミングにおける再配置手法やメモリ内のプログラム実行をフレキシビリティに使用できます．

〔1〕 **オフセット付きモード**　実行アドレスは，プログラム・カウンタと符号拡張された16ビット変位の和で示されます．また，このアドレッシング・モードは1ワードの拡張ワードが必要です．

機　能：EA=(PC)+d

RTL 表記：PC@(d)

モード：111

レジスタ：010

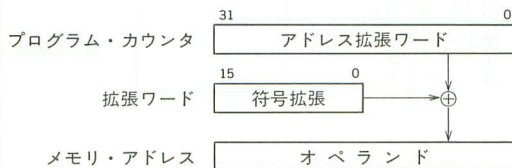



図 5・16

〔2〕 **インデックスとオフセット付きモード**　実行アドレスは，プログラム・カウンタと拡張された8ビット変位整数とインデックス・レジスタの和で示されます．また，プログラム・カウンタの値は拡張ワードのアドレスを示します．このアドレッシング・モードは1ワードの拡張ワードが必要です．

〔例〕 MOVE〈LABEL〉,D0

この命令を実行することにより，アセンブラが自動的にアドレスを計算し，

図 5・17

68000はLABELの付いた$9002番地の内容“4321”をデータ・レジスタD0の下位16ビットにローディングします.

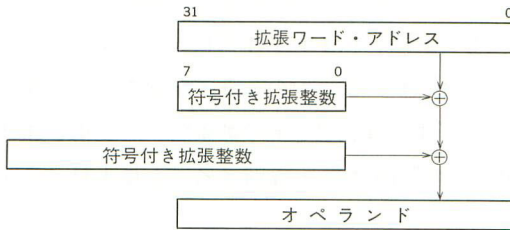



図 5・18

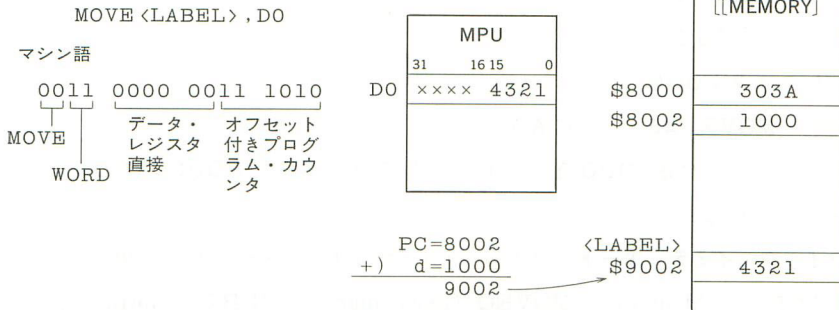



図 5・19 オフセット付きプログラム・カウンタ

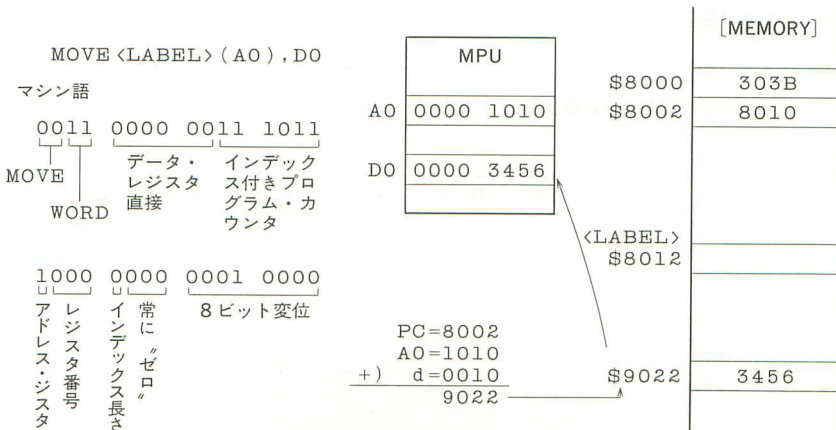



図 5・20 インデックス付きプログラム・カウンタ

5•6　イミディエート・モード

イミディエート・モードは，バイト・オペレーション，ワード・オペレーション，ロング・ワード・オペレーションの3タイプがあり，操作されるデータのサイズに応じて，1ワードまたは2ワードの拡張ワードを用います．また，アセンブラは，RTL 表記の“#”の文字を識別することによりイミディエート・データと判断します．

機　能：オペランドで与えられる．

RTL 表記：#×××

モード：111

レジスタ：000

〔例〕 MOVE　#$1000,A0

ワード・データ$1000をアドレス・レジスタA0に$00001000がローディングされる．

クイック・イミディエート：クイック・イミディエート・モードの使用可能な命令は，ADDQ（add quick），MOVEQ（move quick），SUBQ（subtract quick）の3種類の命令です．

〔例〕 MOVEQ　#$3A,D3

バイト・データの$3Aがデータ・レジスタD3の下位8ビットにD3（データ・レジスタの内容“$0000003A”）にローディングされます．

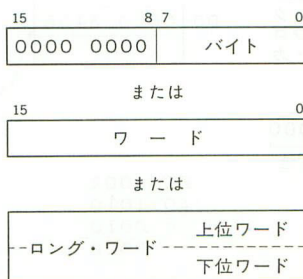



図 5・21

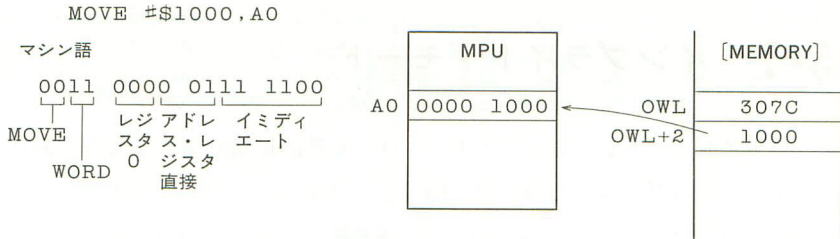



図 5・22 イミディエート・アドレッシング・モード

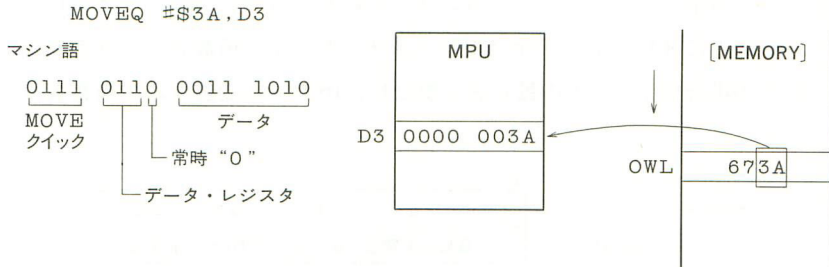



図 5・23 クイック・イミディエート・モード

5・7 インプライド・モード

インプライド・アドレッシング・モードは，暗黙的に指定できる命令があります．つまり，プログラム・カウンタ（PC），システム・スタック・ポインタ（SP），スーパバイザ・スタック・ポインタ（SSP），ユーザ・スタック・ポインタ（USP），ステータス・レジスタ（SR）などです．

ブランチ命令のアドレッシングは，リラティブ・アドレッシングでオペレーション・ワード内に8ビットのディスプレースメント（2の補数表示）があり，これが“0”の場合は1ワードの拡張を必要とし，16ビット変位（2の補数表示）となります．

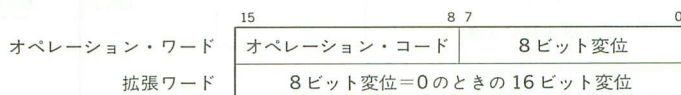



図 5・24　分岐命令フォーマット

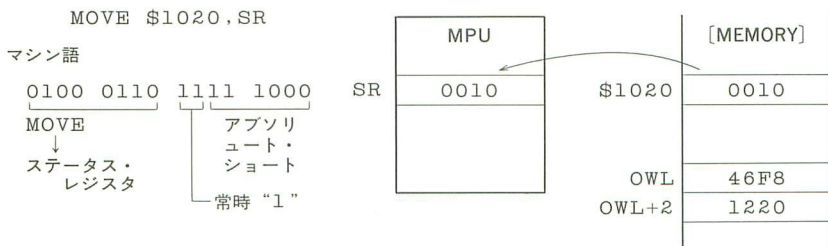



図 5・25　ステータス・レジスタ直接

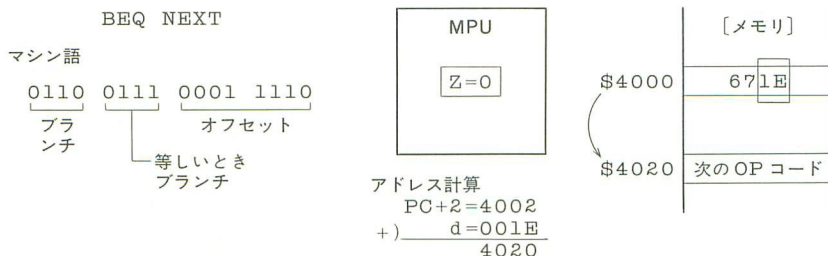



図 5・26　8ビット変位ブランチ（フォワード時）

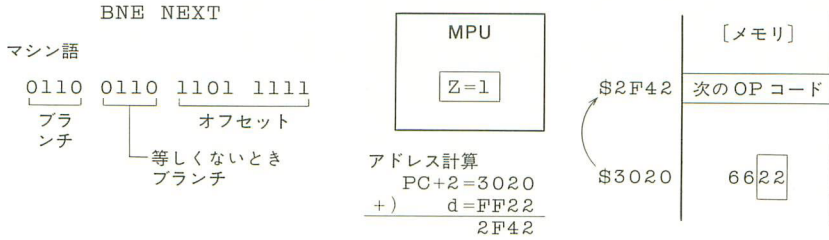



図 5・27 8ビット変位ブランチ（バックワード時）

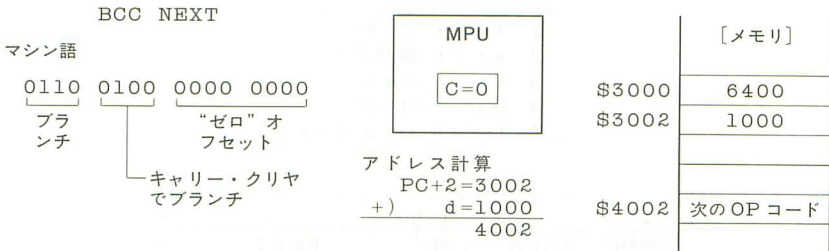



図 5・28 16ビット変位のとき

インプリシット命令一覧表

インストラクション	インプライド・レジスタ
Branch Conditional (Bcc), Branch Always (BRA)	PC
Branch to Subroutine (BSR)	PC, SP
Check Register against Bounds (CHK)	SSP, SR
Test Condition, Decrement and Branch (DBcc)	PC
Signed Divide (DIVS)	SSP, SR
Unsigned Divide (DIVU)	SSP, SR
Jump (JMP)	PC
Jump to Subroutine (JSR)	PC, SP
Link and Allocate (LINK)	SP
Move Condition Codes (MOVE CCR)	SR
Move Status Register (MOVE SR)	SR
Move User Stack Pointer (MOVE USP)	USP
Push Effective Address (PEA)	SP
Return from Exception (RTE)	PC, SP, SR
Return and Restore Condition Codes (RTR)	PC, SP, SR
Return from Subroutine (RTS)	PC, SP
Trap (TRAP)	SSP, SR
Trap on Overflow (TRAPV)	SSP, SR
Unlink (UNLK)	SP

基本的なデータ形式

1. 1ビット(ビット)
2. 4ビット(BCDデジット)
3. 8ビット(バイト)
4. 16ビット(ワード)
5. 32ビット(ロング・ワード)

16 BITS

(MSB) BYTE 0 (LSB)	BYTE 1
BYTE 2	BYTE 3

INTEGERS

(MSB)	(LSB)

WORD

(MSB) HIGH ORDER	
LOW ORDER	(LSB)

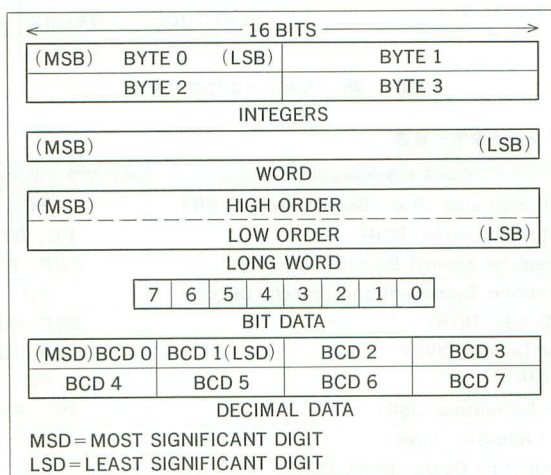
LONG WORD

7	6	5	4	3	2	1	0

BIT DATA

(MSD)BCD 0	BCD 1(LSD)	BCD 2	BCD 3
BCD 4	BCD 5	BCD 6	BCD 7

DECIMAL DATA

MSD=MOST SIGNIFICANT DIGIT
LSD=LEAST SIGNIFICANT DIGIT

データ長とモードの関係

モード ＼ データ・サイズ	バイト	ワード	ロング・ワード
ポストインクリメント	+1	+2	+4
プリデクリメント	-1	-2	-4

6. 例外処理

システムの正常動作のチェックと異常動作時の処理は，システムの信頼性を左右する重要な要素です．68000は，豊富な例外処理機能を備え，この点を十分にカバーすることを説明しています．例外処理の種類と内容や割込みの機能，そしてベクタやアドレスの関係を説明します．

6・1 例 外 処 理

68000の処理状態は，ノーマル処理，停止（ホルト），例外処理の3方法がとられており，そのうちのどれかの処理状態にあります．68000の異常動作の処理方法や割込み処理方法として例外処理（エクセプション・プロセッシング）があり，この方法によって68000の高効率化と高信頼性を得ています．

例外処理には**内部起因による例外**と**外部起因による例外**の2種類があり，内部要因例外とは，アドレス・エラー，不正命令，特権違反などがあり，外部要因例外は，割込み，バス・エラー，リセットなどです．また，例外処理は**3**グループに分類され，**256**種類の定義（例外ベクタ割当て）があり，それぞれ固有の優先度とベクタ番号が割り当てられています（表6・1，表6・2参照）．

プロセッサは，例外処理が生ずると例外処理ルーチンのアドレスをフェッチ（メモリ・ロケーション）してジャンプする（例外ベクタ）．図6・1に示すように，例外ベクタは2ワード長（リセット・ベクタを除く）であり，ベクタ番号は8ビットで表し，このベクタ番号を4倍にして例外ベクタのアドレスを作ります．図6・3に示すようにプロセッサは，8ビットのベクタ番号を24ビットのアドレスに変換します．

［**備考**］ 割込みは，周辺デバイスから出されるプロセッサへの動作要求であり，

表 6・1 例外処理の分類と優先度

グループ	例外種類	処　理
0	リセット バス・エラー アドレス・エラー	実行中の命令はアボートされ，すぐに例外処理を開始する
1	トレース 割込み 不正命令 特権違反	実行中の命令が完結した後，次の命令をフェッチして例外処理を開始する
2	TRAP命令 TRAPV命令 CHK命令 0による割算	ノーマルの命令処理として例外処理に入る

表 6・2　例外ベクタの割当て

ベクタ番号	アドレス			割当て
	10進	16進	参照区分[2]	
0	0	000	SP	リセット：SSPの初期値
	4	004	SP	リセット：PCの初期値
2	8	008	SD	バス・エラー
3	12	00C	SD	アドレス・エラー
4	16	010	SD	不正命令
5	20	014	SD	ゼロによる割算
6	24	018	SD	CHK命令
7	28	01C	SD	TRAPV命令
8	32	020	SD	特権違反
9	36	024	SD	トレース
10	40	028	SD	1010エミュレータ
11	44	02C	SD	1111エミュレータ
12[1]	48	030	SD	(未定義)
13[1]	52	034	SD	(未定義)
14[1]	56	038	SD	(未定義)
15[1]	60	03C	SD	(未定義)
16～23[1]	64	040	SD	(未定義)
	95	05F		—
24	96	060	SD	スプリアス割込み
25	100	064	SD	自動ベクタ割込み(レベル1)
26	104	068	SD	自動ベクタ割込み(レベル2)
27	108	06C	SD	自動ベクタ割込み(レベル3)
28	112	070	SD	自動ベクタ割込み(レベル4)
29	116	074	SD	自動ベクタ割込み(レベル5)
30	120	078	SD	自動ベクタ割込み(レベル6)
31	124	07C	SD	自動ベクタ割込み(レベル7)
32～47	128	080	SD	TRAP命令
	191	0BF		—
48～63[1]	192	0C0	SD	(未定義)
	255	0FF		—
64～255	256	100	SD	ユーザ割込み
	1023	3FF		—

注 1)　ベクタ番号12～23，48～63は，現在使用されていないが，将来使用される可能性があるため，ユーザの周辺デバイスにこれらの番号（アドレス）を割当てできない．

2)　参照区分のSPはスーパバイザ・プログラム，SDはスーパバイザ・データ．

バス・エラーとリセット入力は，アクセス・コントロールとプロセッサのリスタートに使用されます．

ノーマル処理は，命令実行に関するものです．メモリ・レファレンスによって命令およびオペランドをフェッチし，その結果をストアする処理です．

停止（ホルト）処理は，致命的なハードウェア上の異常により発生します．例

図 6・1　例外ベクタのフォーマット

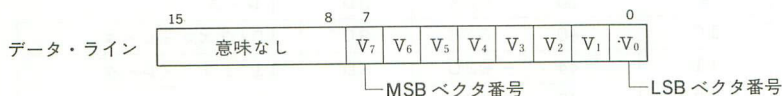



図 6・2　ベクタ番号のフォーマット

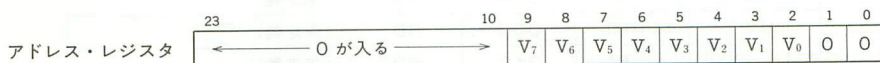



図 6・3　8ビットのベクタ番号から交換されるアドレス

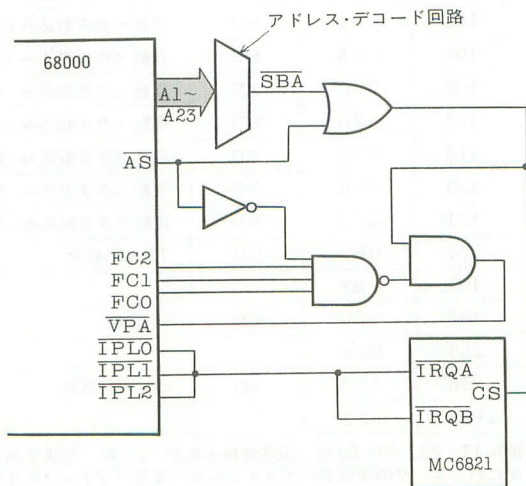



図 6・4　簡単な割込みストラクチャ

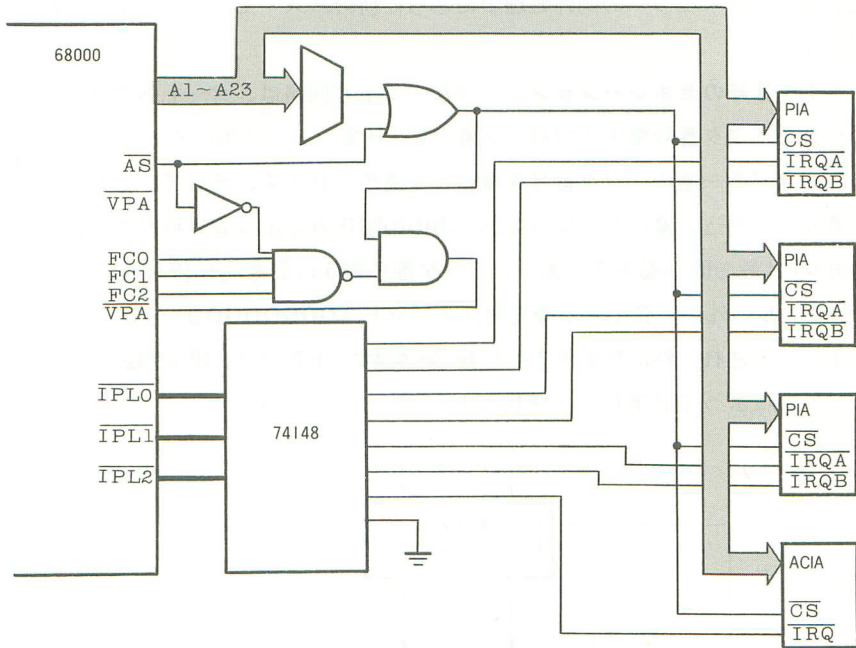



図 6・5 6800周辺マルチ割込み

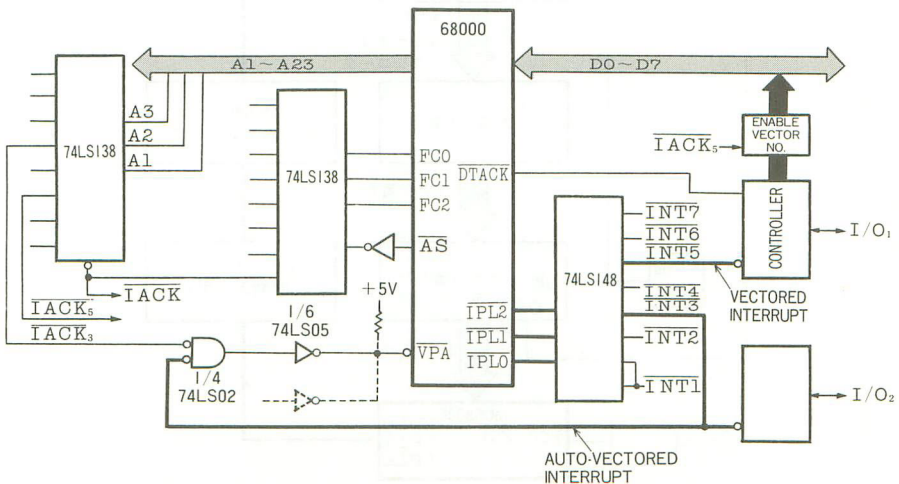



図 6・6 オート・ベクタ割込みとベクタ割込み回路

として，バス・エラーの例外処理中にさらに別のバス・エラーが発生したときなどです．

ベクタ番号のゼネレーション　図6·7 に示す回路は，オートベクタで必ず用いられるベクタ番号発生のブロック図です．192 のユーザ割込みベクタは，64 から 255 までの一連のベクタ番号を参照して決められます．そして，これらのベクタ番号は，割込み優先度に従っておのおのの割込みが振り分けられます．ベクタ番号 **64** は最も低い優先度であり，ベクタ番号 **255** は最も高い優先度です．この 192 レベルの外部で発生する優先度は 8 ビット（00000000～10111111）によって表され，外部発生優先度に 64 のオフセットをかけ，優先割込みエンコーディングによってゼネレートされます．

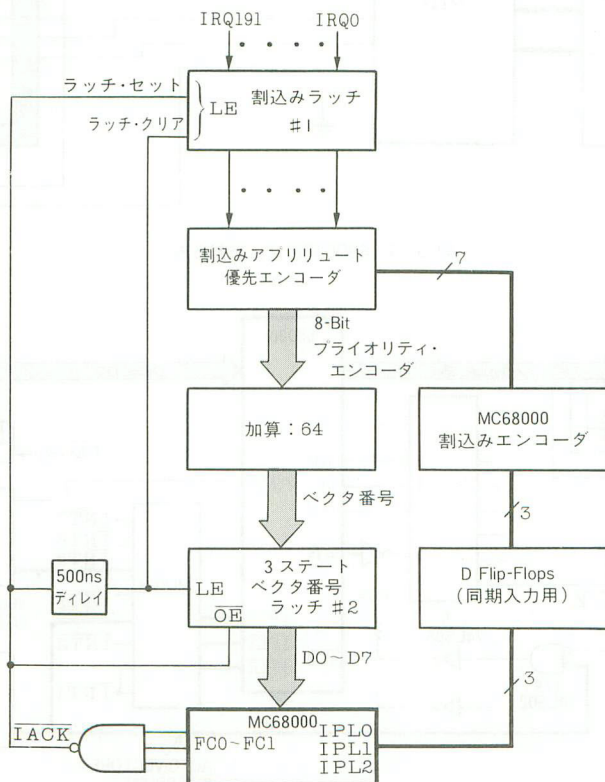



図 6·7　ベクタ番号発生回路ブロック図

図6·9は，ベクタ番号発生回路のスケマティック・ダイアグラムを示します．この図は，192種類の割込みベクタ番号を供給するためのものであり，192本の割込み要求線は，オクタル・ラッチ用のSN74LS373と3〜8のエンコーダSN74LS148を通して8グループに構成され，それぞれ24区分されているスケマティックです．そして，8レベルの優先度とおのおののグループへの割込みは，3ビットのエンコーダ信号線のA0，A1，A2によって割り振られます．

図6·10に示す割込みエンコーディング回路は，割込み処理を行う方法の回路例です．プライオリティ・エンコーダからのすべての**バリッド**割込み入力は，プロセッサの**IPL**ピンで表示されます．そして，プライオリティ・エンコーダが動作状態のときには，プロセッサがIPLピンをサンプルして，現行のプロセッサ・レベルより高いかどうかを判断します．

例えば，システムがレベル2で動作しているときには，レベル3割込みをプライオリティ・エンコーダは保持し，現行命令実行サイクル終了後にレベル3割込みが実行されます．また，レベル3割込みが実行される前にレベル4の割込みが受け付けられた場合は，プライオリティ・エンコーダの出力を短時間でグリッチし，レベル4の出力へと変更させます．

割込み入力は，SN74LS273の8ビット・レジスタにてラッチされ，SN74LS348の8〜3ラインのプライオリティ・エンコーダでもってコンバートされます．フリップ・フロップSN74LS175は，SN74LS273とSN74LS348を通すことによるディレイ・タイムとIPL入力ピンへのインプット・セットアップのためのものです．

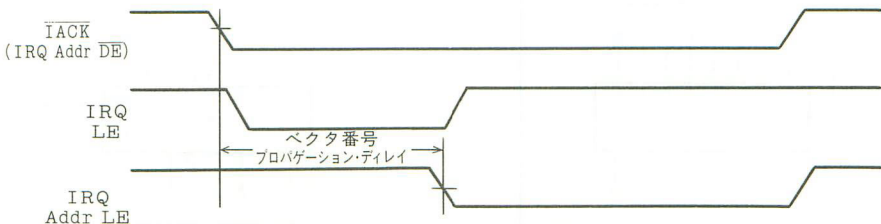
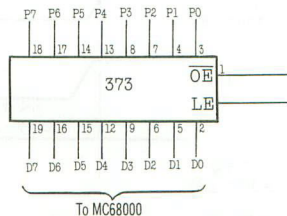
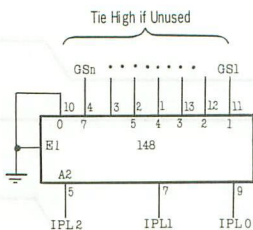
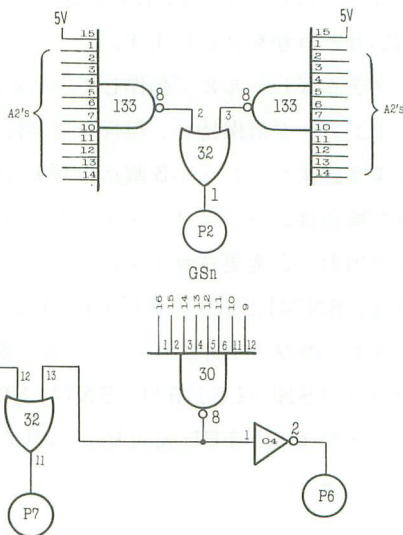
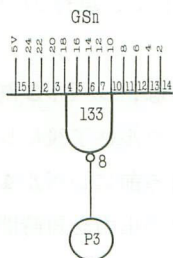
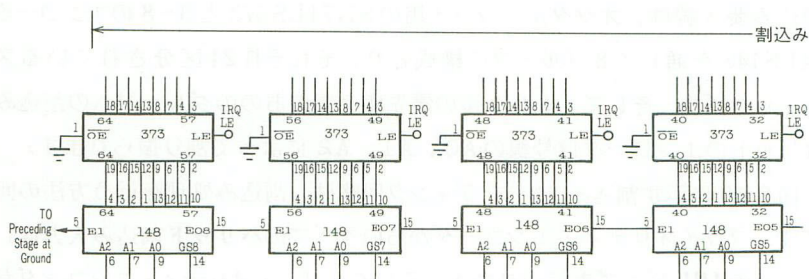
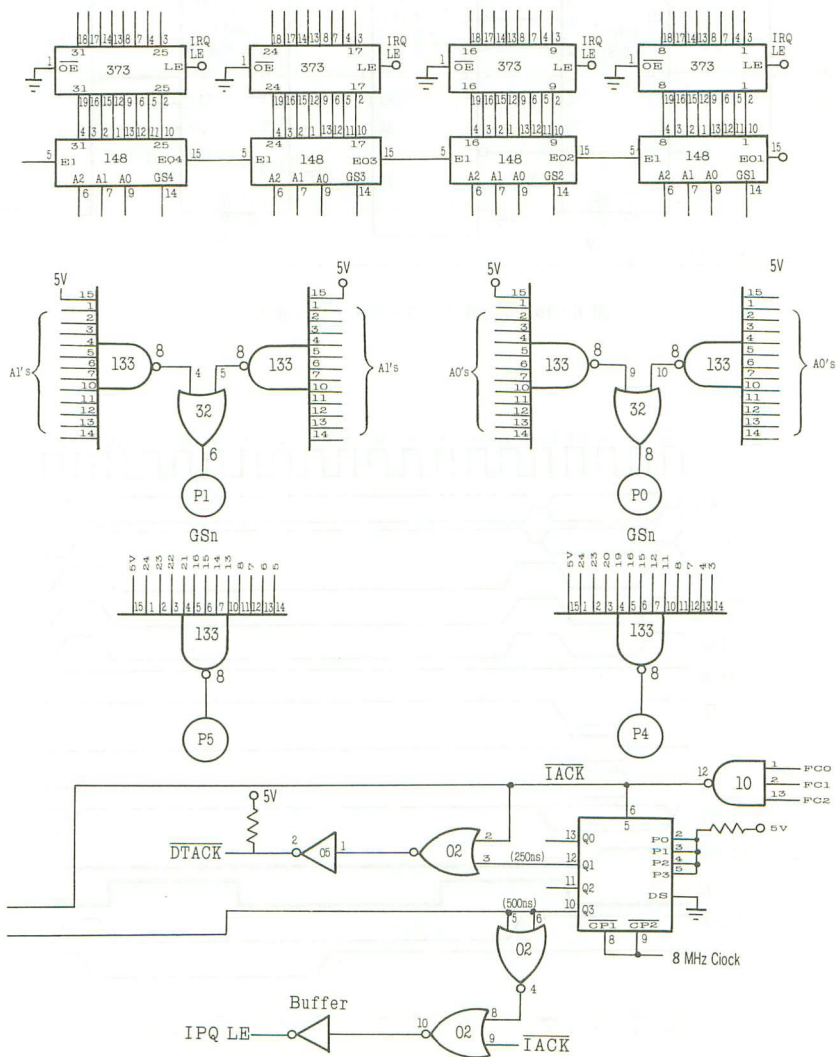



図 6·8　ベクタ番号発生回路のタイミング図

割込み
IRQ LE
OE
373
148
TO Preceding Stage at Ground
E1
EO8
EO7
EO6
EO5
A2 A1 A0 GS8
GS7
GS6
GS5
GSn
5V
133
32
30
04
A2's
P3
P2
P7
P6
Tie High if Unused
GSn
GS1
IPL2
IPL1
IPL0
P7 P6 P5 P4 P3 P2 P1 P0
D7 D6 D5 D4 D3 D2 D1 D0
To MC68000

図 6・9　ベクタ

入力 1～64
IRQ
LE
OE
373
148
E1
EO4
EO3
EO2
EO1
A2
A1
A0
GS4
GS3
GS2
GS1
5V
133
32
A1's
A0's
P1
P0
GSn
P5
P4
IACK
10
FC0
FC1
FC2
DTACK
05
02
(250ns)
(500ns)
Q0
Q1
Q2
Q3
P0
P1
P2
P3
DS
CP1
CP2
8 MHz Ciock
Buffer
IPQ LE

番号発生回路例

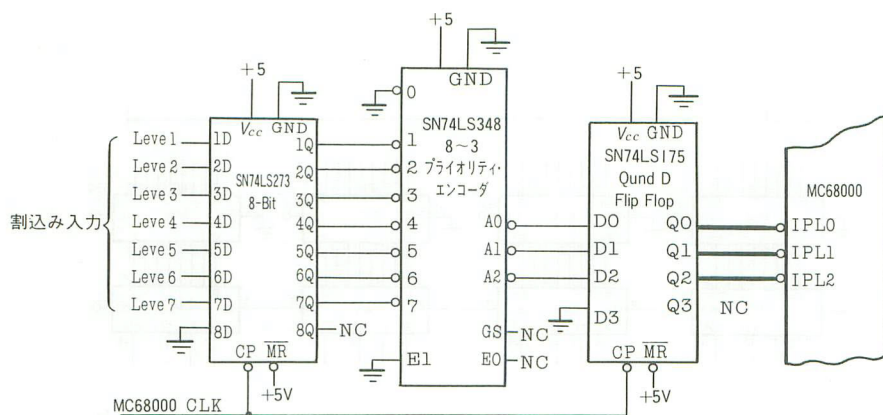



図 6・10 割込みエンコーディング回路例

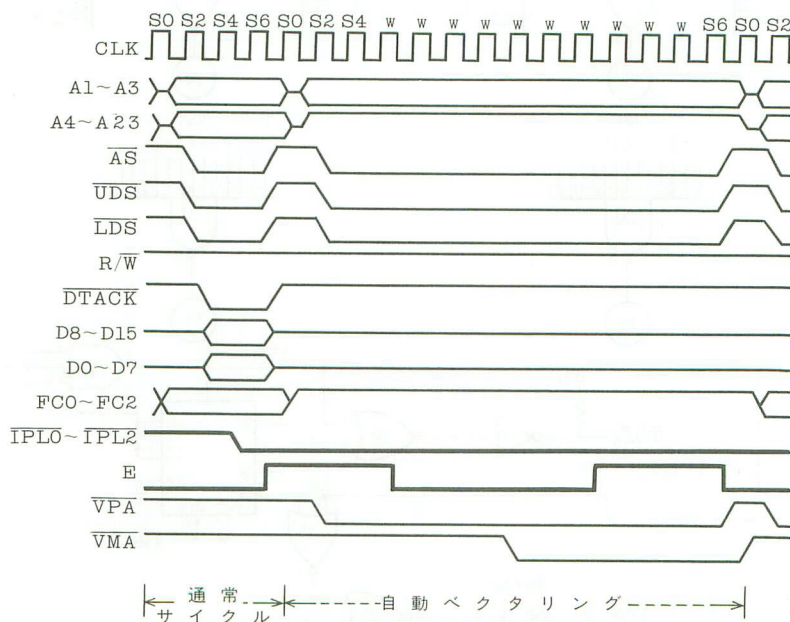



図 6・11 自動ベクタ・タイミング

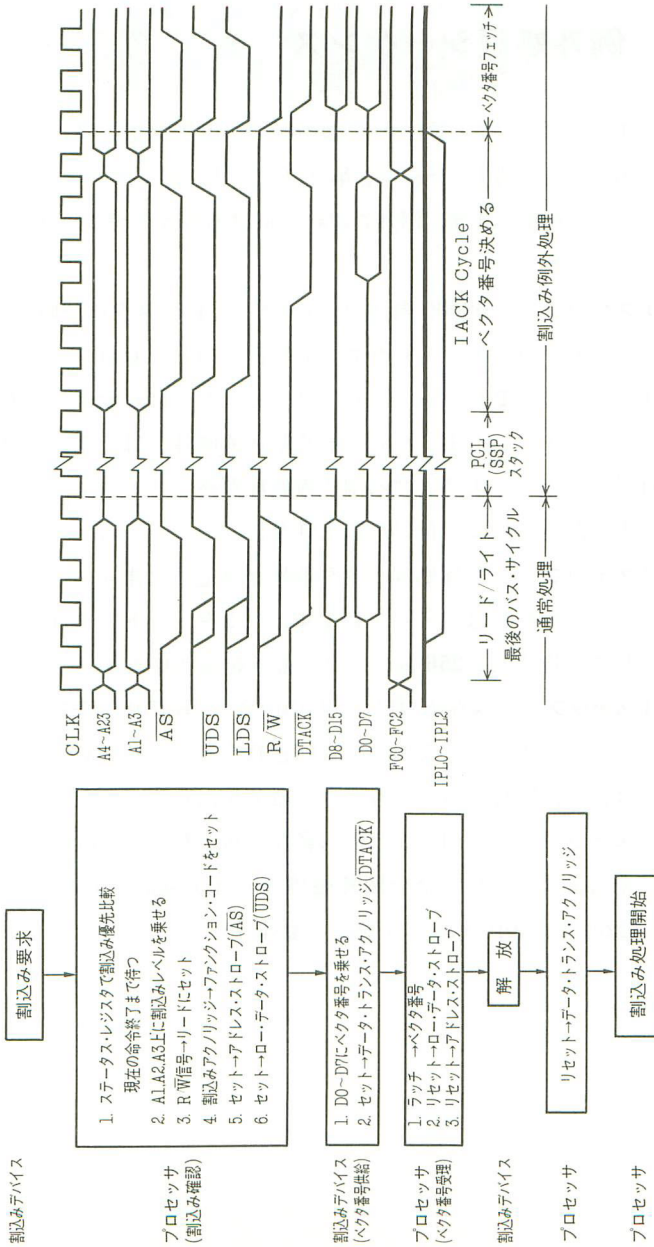



図 6・12 割込み処理タイミング

6・2 例外処理シーケンス

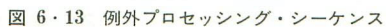
例外処理（エクセプション・プロセッシング）シーケンスは，そのエクセプションの起こる原因により，第1ステップ，第2ステップ，第3ステップ，第4ステップと四つのステップを踏みます．例外処理時における例外処理シーケンスフロ ーを図6・13に示します．

［1］ **第1ステップ**　68000内部のステータス・レジスタへ一時的に待避し，Sビット（スーパバイザ・モード）がセットし，プロセッサはスーパバイザ特権状態になります．また，Tビット（トレース・モード）はリセットされて，例外処理ルーチンに入ります．なお，リセットや割込み例外では，割込み優先度マスクも更新され，現在実行しようとする割込み処理が終了するまでは，処理中の割込み優先度以下の割込みは受け付けられません．

［2］ **第2ステップ**　例外処理ベクタ番号が決定されます．また，割込みでのベクタ番号をプロセッサはフェッチします．ベクタ・アドレスは，ベクタ番号に**4**が乗じられて変換され，**256**個のベクタ番号を識別します．

［3］ **第3ステップ**　現在のプロセッサの内容（SRの値やPCの値）がスーパバイザ・スタック・ポインタを用いて収納されます．収納されたプログラム・カウンタの値は，まだ実行されていない次の命令を指します．

［4］ **第4ステップ**　プロセッサの内容が一新されて，プロセッサはベクタ・テーブルから先頭番地を求めて命令例外処理の実行を開始します．通常はRET命令（RET from Exception）で終了させます．

図 6・13　例外プロセッシング・シーケンス

6・3 バス・エラー

バス・エラーは，外部デバイスとハンドシェイクを行うことを前提として，外部回路はアドレス・ストローブ（$\overline{\mathrm{AS}}$）とデータ転送アクノリッジ（$\overline{\mathrm{DTAK}}$）間の時間を測定し，エラーか正常かを判断し，バス・エラー（$\overline{\mathrm{BEER}}$）信号を発生させます．プロセッサは，バス・エラー信号を受信すると，バス・エラー例外シーケンスを開始するか，バス・サイクルを再実行（リラン）するかを決めます．また，バス・エラーやアドレス・エラーやリセットの例外処理中にバス・エラーが発生した場合は，プロセッサはホルトになり，メモリ内容を保護するためにすべての処理を中止します．なお，ホルトしたプロセッサの解除は，外部からのリセット入力信号によって行います．表6・3にバス・エラー信号と機能の関係を示しました．

〔1〕 **バス・エラー例外シーケンス**　プロセッサがバス・エラー信号（$\overline{\mathrm{BEER}}$）を受信し，ホルト（$\overline{\mathrm{HALT}}$）がセットされていないときにバス・エラー例外シーケンスに入ります．

処理手順 1） プログラム・カウンタとステータス・レジスタを収納．
2） エラー情報を収納．
3） バス・エラーのベクタ開始テーブルを読む．
4） バス・エラーのルーチンを実行する．
（バス・エラーのベクタ番号は2でアドレスは008）

〔2〕 **バス・サイクルのリラン**　プロセッサがバス・エラー信号（$\overline{\mathrm{BEER}}$）を受信し，ホルト（$\overline{\mathrm{HALT}}$）が外部デバイスによってセットされているとき，プ

表 6・3 バス・エラー信号と機能の関係

$\overline{\mathrm{BERR}}$	$\overline{\mathrm{HALT}}$	機　能
0	0	現在のサイクルを再実行
0	1	例外処理
1	0	シングル・バス・サイクル・オペレーション
1	1	ノーマル・オペレーション

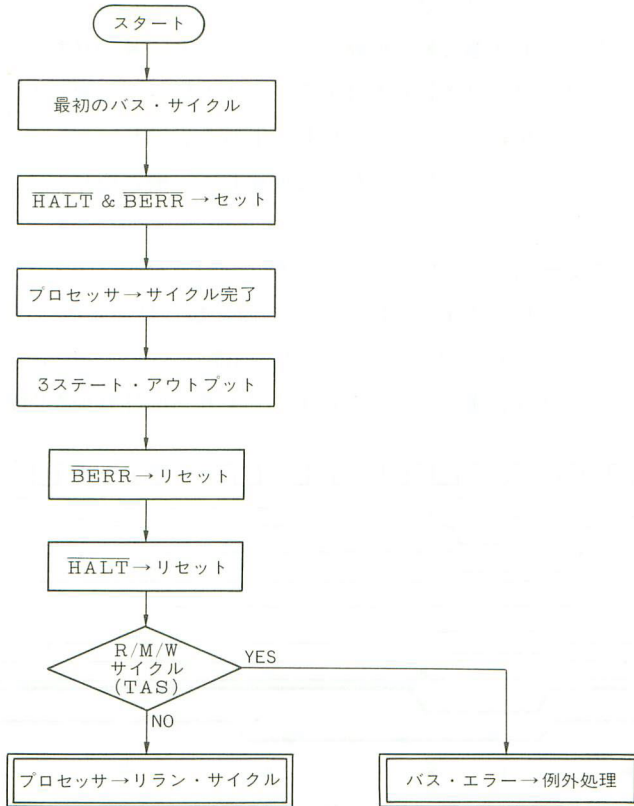



図 6・14　リラン・サイクル・フロー

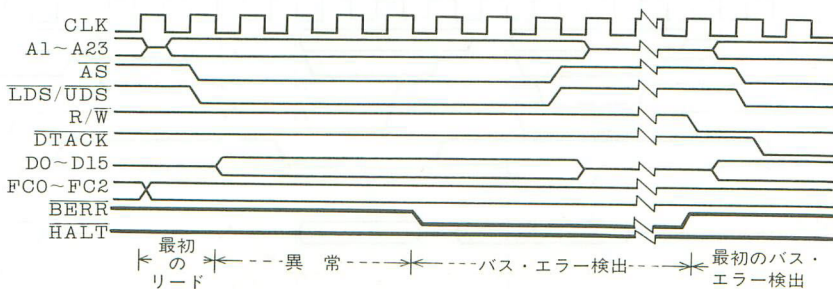



図 6・15　バス・エラーのタイミング

ロセッサはリラン・シーケンスに入ります．

プロセッサがホルト状態で外部回路によってホルト信号がリセットされるまでは，次のバス・サイクルは実行されません．ホルト信号がリセットされると，プロセッサは，エラーを起こしたときと同じアドレス，同じファンクション・コード，同じデータ，同じコントロール信号を使用して，前と同じバス・サイクルを再実行（リラン）します．

アドレス・エラー例外は，ワード・アクセスやロング・ワード・アクセス時に起きたり，奇数アドレスのインストラクション時に起こります．アドレス・エラーが生ずると，バス・サイクルはアボードされ，内部バス・エラーを発生させます．そして，プロセッサは，現在の処理を終了させてから例外処理を開始します．例

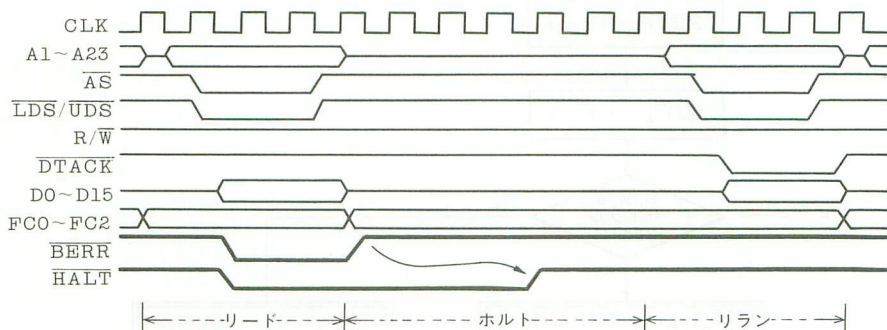



図 6・16　リラン・バス・サイクルのタイミング

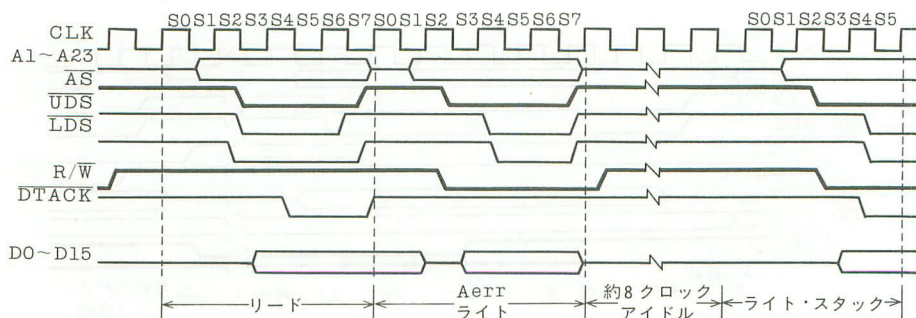



図 6・17　アドレス・エラーのタイミング

外処理開始後のシーケンスは，バス・エラーと同様に実行されます．ただし，アドレス・エラー・ベクタを参照し，ベクタ番号を格納することを除きます．また，アドレス・エラーがバス・エラー例外処理中に起こったときには，プロセッサはホルトになります．

図6·17は，アドレス・エラー時の例外処理によってフォローされるショート・バス・サイクルの実行タイミング図です．

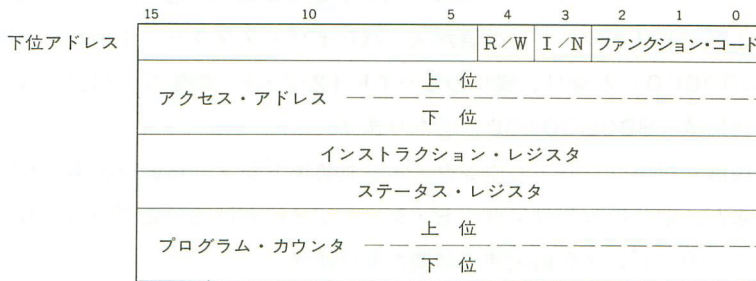
	15　　10　　5	4	3	2　1　0
下位アドレス		R／W	I／N	ファンクション・コード
アクセス・アドレス	上 位			
	下 位			
	インストラクション・レジスタ			
	ステータス・レジスタ			
プログラム・カウンタ	上 位			
	下 位			

R／W：ライト＝0
　　　リード＝1
I／N：インストラクション＝0

図 6・18　バス・アドレス・エラー・エクセプションのためのスーパバイザ・スタック配列

6・4 リセットとトラップ

外部要因によってリセットが発生すると，進行中の処理はすべてクリアされ，Sビットがセットし，Tビットはリセットします．そして割込み優先度は最も高い“7”であり，プロセッサは強制的にスーパバイザ状態にセットされます．図6・19に示すように，例外ベクタ領域；ベクタ番号0におけるベクタ・アドレスは，最初の4バイト（2ワード）の値がスーパバイザ・スタック・ポインタの初期値（$00200000）となり，残りの4バイト（2ワード）の値がプログラム・カウンタの初期値（$00E00100）となります．

その後は，プログラム・カウンタで示されるアドレスから命令の実行を開始します．また，電源投入やリスタート・シーケンスにおけるプログラム・カウンタの初期の先頭アドレスを指定する必要があります．

ソフトウェアによるRESET命令では，リセット・ベクタをローディングせずにリセット・ラインをセットして，外部デバイスをリセットします．リセット後は，RESET命令の次の命令から実行されてゆきます．

トラップは，命令に起因する例外です．プロセッサが命令実行中に異常な状態を認めたか，ノーマル動作としてトラップを発生させるような命令を使用して発生するものの2とおりがあります．ステータス・レジスタとプログラム・カウンタはスーパバイザ・スタックに退避され，例外ベクタに格納されているベクタ・アドレスにより命令の実行を開始します．

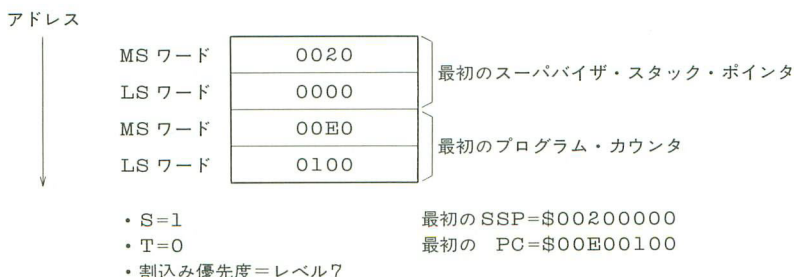



図 6・19　リセット例外ベクタ

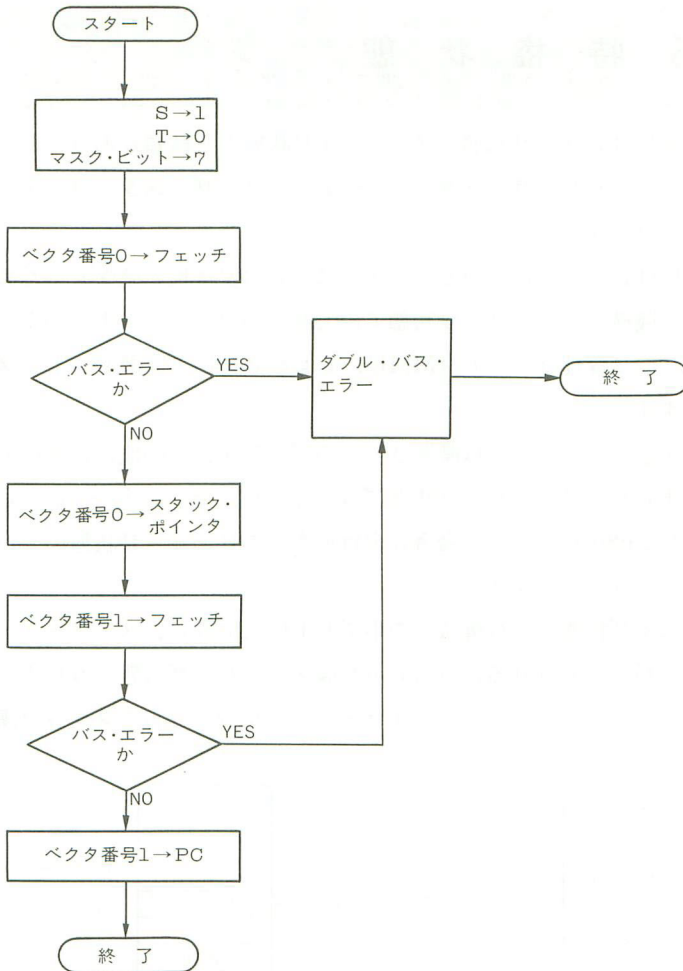



図 6・20　リセット例外プロセッシング・フロー

表 6・4　トラップに関する命令と機能

命　令	機　能
TRAPV CHK	エラー・チェック用 エラーが発生したときだけトラップが発生する
DIVS DIVU	エラー・チェック用 エラーが発生したときだけトラップが発生する
TRAP0 ～ TRAP15	常に例外処理を発生させる システム・コール，ブレーク・ポイント，エミュレートなどに用いる

6・5 特 権 状 態

特権状態には，ユーザ状態とスーパバイザ状態の2種類があり，命令レファレンスには，スーパバイザ・スタック（SSP）とユーザ・スタック・ポインタの2とおりがあります．

特権状態は，マイクロプロセッサ・システムの安全性を確保するためのものであり，特権機構におけるユーザ状態での大部分のプログラム実行の安全性を持たせ，外部メモリ管理ユニット（MMU）によるアクセス制御やアドレス変換などに使用します．

ユーザ状態でのアクセスは制御されるので，システムの他の領域へのアクセスは許されません．スーパバイザ状態では，オペレーティング・システム（OS）の管轄のもとに68000の持つ全命令が実行可能であり，ユーザ状態のプログラム管理タスクの実行ができます．

スーパバイザ状態は，特権状態の中でも上位に位置し，ステータス・レジスタのSビットがセットされるとプロセッサはスーパバイザ状態になります．このときのバス・サイクルは，スーパバイザ・レファレンスとしてスタック操作をスー

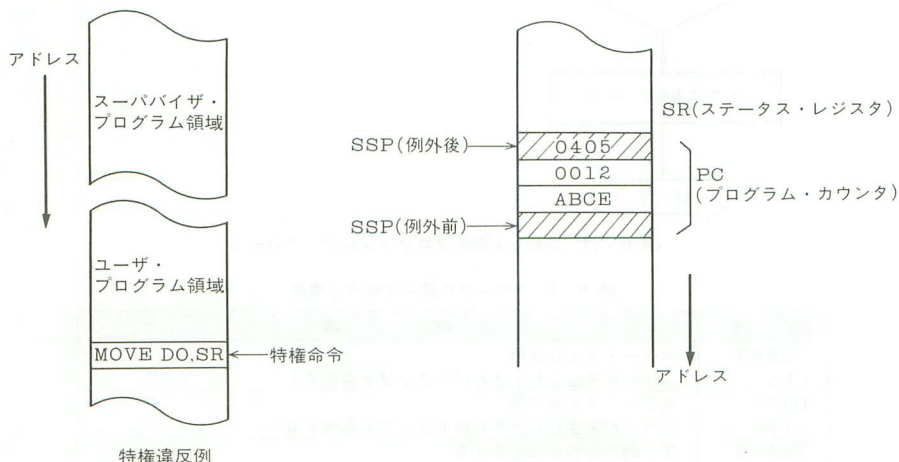



図 6・21 例外処理のスタック方法

パバイザ・スタック・ポインタ（SSP）にします．

ユーザ状態は特権状態の中でも下位に位置し，ステータス・レジスタのSビットがリセットされていると，プロセッサはユーザ状態で命令を実行します．バス・サイクルは，ユーザ状態レファレンスとして，スタック操作とシステム・スタックポインタ，またはユーザ・スタック・ポインタ（USP）にします．なお，システムに対して重要な影響を与える命令は，ユーザ状態では実行できません．まだ，デバッギング・プログラムに用いるユーザ・スタック・ポインタへの移動命令（MOVE to USP）や，ユーザ・スタック・ポインタからの移動命令（MOVE from USP）もユーザ状態では実行できません．

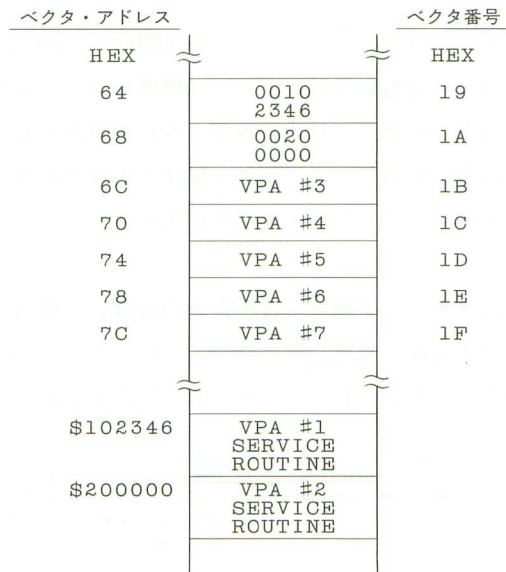



図 6・22　例外ベクタの例

6・6 特権違反とトレース

特権命令は，システムを安全に動作させるための命令であり，次の八つの命令があります．STOP，RESET，RTE，MOVE USP，MOVE to SR，AND Imediate to SR，EOR Imediate to SR，OR Imediate to SR です．

特権違反時の例外処理は，命令をデコードした後に，プロセッサが特権違反かどうかを判断して，例外処理の開始を行います．また，命令のオペレーション・ワードのビット・パターンが正規なものと異なったビット・パターンの命令を**違法命令**といいます．

トレース・モードは，ステータス・レジスタ（SR）のTビットをセットすることによって，現在の命令実行終了後にトレース例外の割込みを起こし，ユーザ・モードのプログラムを1命令ごとに実行をモニタしたり解析したりすることができます．そして，命令実行後モニタへ制御を戻すことができます．Tビットがリセットされているときは，トレース・モードに入らず，ノーマルな命令を次々と実行します．

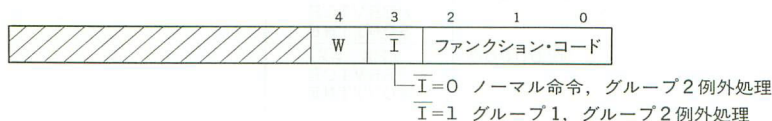
割込み優先度による命令の不履行や違法命令の命令，特権命令などは，トレース・モードには入りません．また，例外時や，アドレス・バス・エラーなどによって命令が中断されたときも，トレース・モードはできません．

スーパ・ステータス・ワード
上位バス・アドレス
下位バス・アドレス
インストラクション・レジスタ
ステータス・ワード
上位プログラム・カウンタ
下位プログラム・カウンタ

図 6・23 違法アドレスとバス・エラーのスタッキング

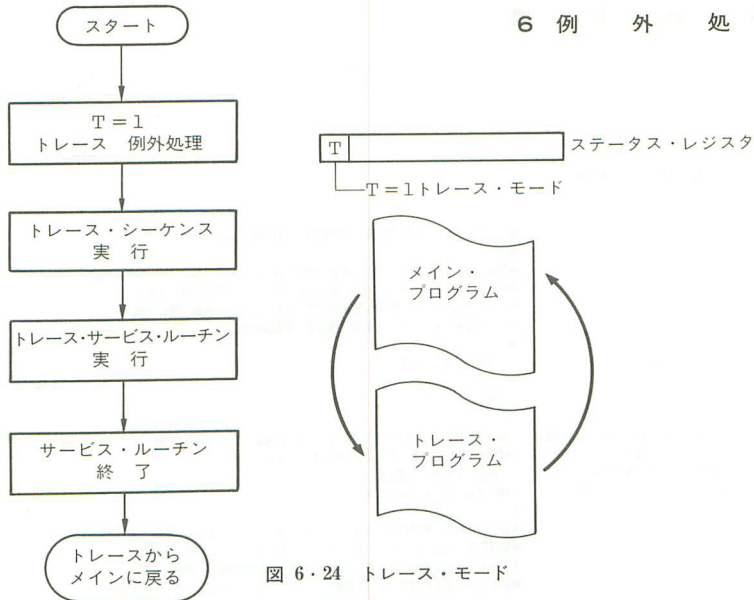



図 6・24 トレース・モード

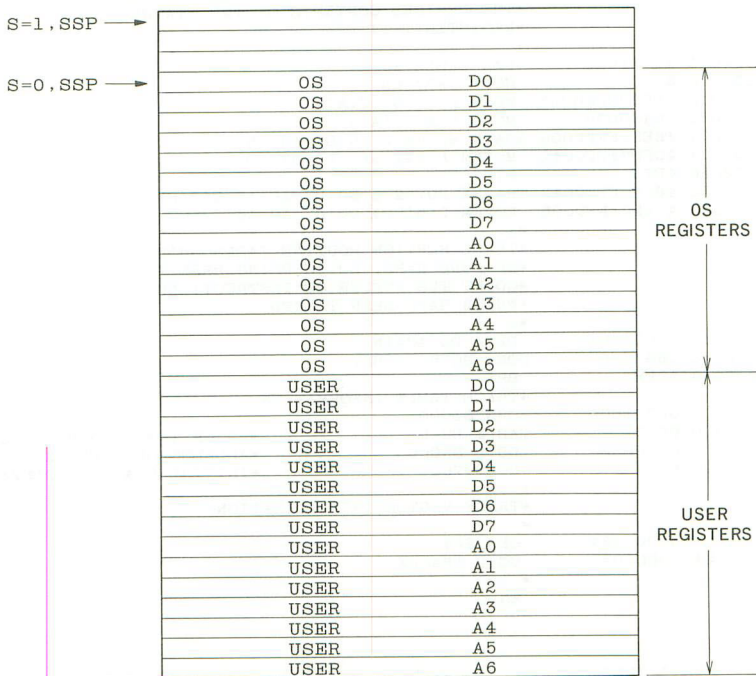
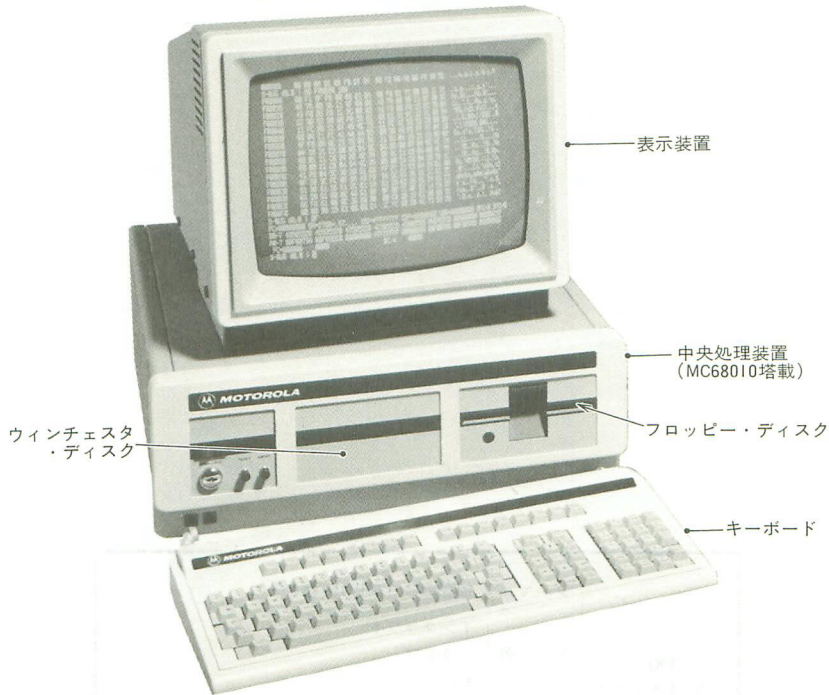



図 6・25 トレース・ストレージ

```
TRACE:1  MC68000 ASM

  10                            *  TRACING THE USER PROGRAM
  20                            *
  30                            *THIS ROUTINE TRACES A SPECIFIED NUMBER
  40                            *OF INSRUCTIONS IN A USER PROGRAM.
  50                            *INSTRUCTIONS EXECUTED IN THE SUPER-
  51                            *VISOR MODE ARE NOT TRACED.
  60                            *
  70         00002000            RORG $2000
  71                            *
  72                            *ENTER OS;S=1
  73                            *
  80 002000 48EF7FFF0042         MOVEM.L D0-D7/A0-A6,$42(SP)
  90 002006 4CEF7FFF0006         MOVEM.L 6(SP),D0-D7/A0-A6
  91 00200C 43FA0024             LEA TRACEX,A1
  92 002010 21C90024             MOVE.L A1,$24
 100                            *
 110                            *DO ANY REQUIRED OS TASKS. A TASK WHICH
 120                            *MUST BE DONE IS TO INITIALIZE THE
 130                            *TRACE TABLE WITH THE NUMBER OF IN-
 140                            *STRUCTIONS TO BE TRACED, THE ADDRESS
 150                            *FROM WHICH TRACING IS TO BEGIN, AND
 160                            *THE DESIRED CONTENTS OF THE STATUS
 170                            *REGISTER(T=1).
 180                            *
 190 002014 41F81000             LEA TABLE,A0
 200 002018 3010                 MOVE (A0),D0
 210 00201A 2F6800020002         MOVE.L 2(A0),2(A7)
 220 002020 3EA80006             MOVE 6(A0),(A7)
 230 002024 48EF7FFF0006        AGAIN MOVEM.L D0-D7/A0-A6,6(SP)
 240 00202A 4CEF7FFF0042         MOVEM.L $42(SP),D0-D7/A0-A6
 250 002030 4E73                 RTE
 260 002032 48EF7FFF0042        TRACEX MOVEM.L D0-D7/A0-A6,$42(SP)
 270 002038 4CEF7FFF0006         MOVEM.L 6(SP),D0-D7/A0-A6
 280                            *
 290                            *TRACE SERVICE ROUTINE TASKS SHOULD
 300                            *BE DONE HERE. D0 IS BEING USED TO
 310                            *COUNT THE NUMBER OF INSTRUCTIONS
 320                            *WHICH HAVE BEEN TRACED.
 330                            *
 340 00203E 51C8FFE4             DBRA D0,AGAIN
 350 002042 08970007            DONE BCLR 7,(SP)
 360 002046 60FE                 BRA *
 361                            *TRACE TABLE INFORMATION
 370         00001000            ORG $1000
 380 001000 0005                TABLE DC 5              * # OF INST. TO BE TRACED
 390 001002 00005000             DC.L $5000             *ADDRESS OF USER PROGRAM
 400 001006 8700                 DC $8700               *INITIHL STATUS REGISTER
 480                            *
 481                            *TRACE VECTOR INITIALIZATION
 482                            *
 490         00000024            ORG $24
 500 000024 00002032             DC.L TRACEX
 501                            *
 510                             END
 520
```

図 6・26 トレース・ユーザ・プログラム

VME/10システム基本構成

特 長

1. アプリケーション開発装置として使用できる.
2. OEM 用組み込み機器として使用できる.
3. VME モジュールと I/O モジュール用のカード・ケージを備えている.
4. 8/16/32 ビット・プロセッサを中心とする機器のハードウェアとソフトウェアの開発をサポート.
5. 8/16/32 ビットのエミュレータ・ステーションと連動することができる.
6. VERSAdos オペレーティングシステムによるサポート.
7. UNIX オペレーティング・システム (SYSTEM V/68) もサポート.
8. システム価格が低廉 (EXORmacs の 1/4).

VME/10 マイクロコンピュータ・システム (その1) 〈その2は p. 131〉

- 68000 処理ステート
  - ノーマル
    - 命令実行（STOP 含む）
  - 例　外
    - インタラプト
    - トラップ
    - トレース
  - ホルト
    - ハードウェア・ホルト
    - ダブル・バス・エラー
    - ダブル・イリーガル
    - アドレス・エラー

- 例外処理
  - 外部要因
    - 割込み（レベル1～7）
    - バス・エラー
    - リセット
  - 内部要因
    - アドレス・エラー
    - トレース
    - TRAP, TRAPV, CHK, DIV 命令
    - 違法命令
    - 特権違反
    - バウンダリ・エラー

7. 命令セット

MC68000の命令は，設計の段階で使用頻度や命令の機械語へのコード化が配慮されています．そして，機能の高い命令セットや例外における退避や復帰の処理方法が強化され，第5章の豊富なアドレッシング・モードとの関連やレジスタとの組合せによって多様な使用方法が可能です．

MC68000の命令を機能別に9個のグループに分け，その代表的な命令の機能を説明します．

7・1　命令形式と命令タイプ

〔1〕 **命令フォーマット**　68000の命令は，1ワードから5ワードの長さがあります．それぞれの命令の長さと実行される操作は，1ワードのオペレーション・ワードとこれに続くイミディエート・オペランドの2ワード，または，オペレーション・ワードの実行アドレス・モードで指定されたオペランドの2ワードで，合計5ワードの命令が最長のものです．また，68000のバイナリ・パターンは，このオペレーション・ワード内に含まれます（図7・1，図7・2参照）．

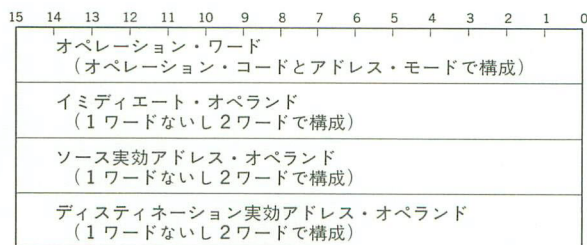
オペレーション・ワードとオペランドは，偶数番地にアドレスしますので注意してください（奇数番地時にはアドレス・エラー・エクセプションが起きます）．

15　14　13　12　11　10　9　8　7　6　5　4　3　2　1　0
オペレーション・ワード （オペレーション・コードとアドレス・モードで構成）
イミディエート・オペランド （1ワードないし2ワードで構成）
ソース実効アドレス・オペランド （1ワードないし2ワードで構成）
ディスティネーション実効アドレス・オペランド （1ワードないし2ワードで構成）

図 7・1　命令フォーマット

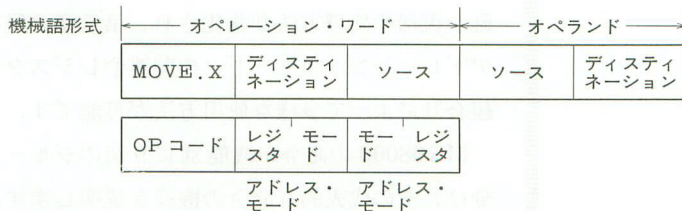



図 7・2　MOVE命令の例

〔2〕 **命令タイプ**　68000の基本命令タイプには56種類のタイプがあります．そのうちのバリエーションのタイプとして8種類があります（表7·1，表7·3参照）．

表 7·1　68000——56の基本命令セット——

ニーモニック	内　　容
ABCD	拡張付きアド・デシマル
ADD	アド
AND	ロジカル・アンド
ASL	算術シフト・レフト
ASR	算術シフト・ライト
BCC	ブランチ・コンディション
BCHG	ビット・テストと変更
BCLR	ビット・テストとクリア
BRA	常時ブランチ
BEST	ビット・テストとセット
BSR	サブルーチンへのブランチ
BTST	ビット・テスト
CHK	レジスタ限界チェック
CLR	オペランド・クリア
CMP	コンペア
DBCC	コンディション・テスト，デクリメントとブランチ
DIVS	符号付き割算
DIVU	符号なし割算
EOR	エクスクルーシブOR
EXG	レジスタ変更
EXT	符号付き拡張
JMP	ジャンプ
SSR	サブルーチンへジャンプ
LEA	実効アドレス・ロード
LINK	スタック・リンク
LSL	ロジカル・シフト・レフト
LSR	ロジカル・シフト・ライト
MOVE	ムーブ
MOVEM	ムーブ・マルチ・レジスタ
MOVEP	ムーブ・ペリフェラル・データ
MULS	符号付きマルチプル
MULU	符号なしマルチプル
NBCD	拡張付きネゲート・デシマル
NEG	ネゲート
NOP	ノー・オペレーション
NOT	1の補数
OR	ロジカルOR
PEA	実効アドレス・プッシュ
RESET	外部デバイス・リセット
ROL	拡張なしのローテート・レフト
ROR	拡張なしのローテート・ライト
ROXL	拡張付きローテート・レフト
ROXR	拡張付きローテート・ライト
RTE	例外からのリターン
RTR	リターンと再格納
RTS	サブルーチンからのリターン
SBCD	拡張付きデシマル割算
SCC	コンディション・セット
STOP	ストップ
SUB	割算
SWAP	レジスタ半分スワップ
TAS	テストとオペランド・セット
TRAP	トラップ
TRAPV	オーバフローにおけるトラップ
TST	テスト
UNLK	アンリンク

7・2 データ移動命令

データ移動命令は，表7・2に示すように汎用レジスタやメモリのデータなどを転送するのに用います．

MOVE（ムーブ・データ・フロム・ソース・トウ・ディスティネーション）のオペレーション・サイズは，バイト，ワード，ロング・ワード長のデータを指定でき，データ・レジスタからデータ・レジスタへ，メモリとデータ・レジスタ間や，メモリからメモリへ転送します．また，データは転送時にチェックし，コンディションコードをセットします．

MOVEM（ムーブ・マルチプル・レジスタ）は，レジスタ・リストで指定した複数のレジスタにより，実効アドレスで指定したところから連続したメモリの位置へ転送したり，転送されたりします（3・3節「スタックの構成」参照）．

MOVEP（ムーブ・ペリフェラル）は，データ・レジスタと1バイトおきのメモリの間でデータ（バイト）転送を行います．データ・レジスタの偶数のアドレ

表 7・2 データ移動命令

命　令	オペランド・サイズ	機　能	内　容
EXG	32	Rx↔Ry	レジスタ交換
LEA	32	EA→An	実行アドレス・ロード
LINK	—	An→SP@- SP→An SP+d→SP	リンクと割付け
MOVE	8, 16, 32	(EA)s→EAd	ソースから行先へデータ移動
MOVEM	16, 32	(EA)→An, Dn An, Dn→EA	複数レジスタ移動
MOVEP	16, 32	(EA)→Dn Dn→EA	周辺データ移動
MOVEQ	8	♯Imm→Dn	クイック移動
PEA	32	EA→SP@-	実行アドレス格納
SWAP	32	Dn〔31:16〕↔Dn〔15:0〕	レジスタ半分スワップ
UNLK	—	An→SP SP@+→An	リンク解除

S：ソース，d：ディスティネーション，〔 〕：ビット番号

スのときはデータ・バスの上半分で転送され，奇数のアドレスのときはデータ・バスの下半分を用いて転送します．

MOVEQ（ムーブ・クイック）は，8ビットのイミディエート・データをディスティネーションのデータ・レジスタに移します．データは，32ビットのロング・ワードに符号拡張されて，データ・レジスタにロードされます．

EXG（エクスチェンジ・レジスタ）は，常にロング・ワード・オペレーションで二つのレジスタ内容を交換します．

表7·3　8タイプの特定操作のバリエーション型

命令タイプ	バリエーション	内　容
ADD	ADD ADDA ADDQ ADDI ADDX	加　算 アドレス加算 クイック加算 イミディエイト加算 拡張付き加算
AND	AND ANDI	ロジカル・アンド イミディエート・アンド
CMP	CMP CMPA CMPM CMPI	比　較 アドレス比較 メモリ比較 イミディエート比較
EOR	EOR EORI	エクスクルーシブOR イミディエート・エクスクルーシブOR
MOVE	MOVE MOVEA MOVEQ MOVE from SR MOVE to SR MOVE to CCR MOVE to USP	転　送 アドレス転送 クイック転送 ステータス・レジスタからの転送 ステータス・レジスタへの転送 コンディション・コード・レジスタに転送 ユーザ・スタック・ポインタに転送
NEG	NEG NEGX	ネゲート 拡張付きネゲート
OR	OR ORI	ロジカルOR イミディエートOR
SUB	SUB SUBA SUBI SUBQ SUBX	割　算 アドレス割算 イミディエート割算 クイック割算 拡張付き割算

LEA（ロード・イフェクティブ・アドレス）は，ソース・オペランドの実効アドレスを指定したアドレス・レジスタにロードします．

PEA（プッシュ・イフェクティブ・アドレス）は，スーパバイザ・スタックかユーザ・スタックかのいずれかに実行アドレスが計算されて，スタックにプッシュされます．

SWAP（スワップ・レジスタ・ハーフ）は，データ・レジスタの上位16ビットと下位16ビットの内容を入れ替えることができます．

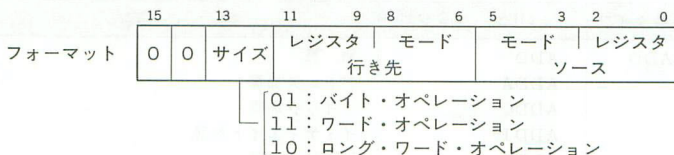
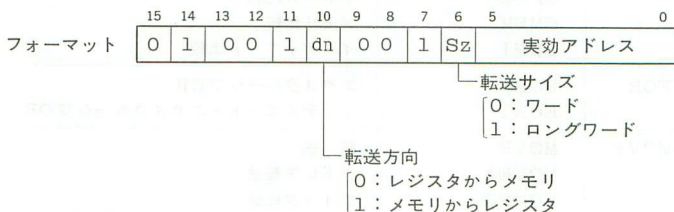
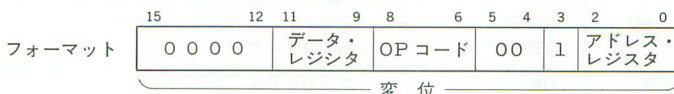
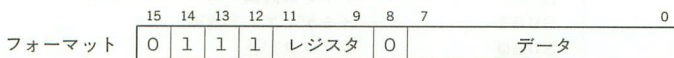



図 7·3 (1) データ移動命令フォーマット

EXG：Rx↔Ry
アセンブラ：EXG Rx,Ry

フォーマット

15	13	11	8	7 3	2 0
11	00	Rx レジスタ	1	OP モード	Ry レジスタ

LEA：ディスティネーション→An
アセンブラ：LEA 〈ea〉, An

フォーマット

15	14	13 12	11 9	8 6	5 0
0	1	00	レジスタ	111	実効アドレス

PEA：ディスティネーション→SP@-
アセンブラ：PEA 〈ea〉

フォーマット

15	14	13	11	10	6	5 0
0	1	00	1	0000	1	実効アドレス

SWAP：レジスタ［31·16］↔レジスタ［15·0］
アセンブラ：SWAP Dn

フォーマット

15	14	13	11	10	6	5	2 0
0	1	00	1	0000	1	000	レジスタ

図 7·3 (2) データ移動フォーマット

MOVE D0,D1	データ・レジスタD0の下位ワードをD1の下位ワードに転送
MOVE.L $01000,$02000	$1000番地のロングワードを$2000番地に転送
MOVE.W #'A',3000	値'A'00を$3000番地へ転送
MOVE $1000,A1	$1000番地のワードをアドレス・レジスタA1の下位ワードへ転送

図 7·4 MOVE 命令の例

```
BLMOVE:1  MC68000 ASM

  1                     *  BLOCK MOVE
  2                     *
  3                     *MOVE A BLOCK OF DATA FROM
  4                     *SOURCE TO DESTINATION. D0 CONTAINS
  5                     *THE NUMBER OF LONG WORDS-1
  6                     *TO BE MOVED. A0 CONTAINS THE SOURCE
  7                     *ADDRESS AND A1 CONTAINS THE
  8                     *DESTINATION ADDRESS.
  9                     *
 10        00001000      ORG $1000
 11 001000 22D8         AGAIN MOVE.L (A0)+,(A1)+
 12 001002 51C8FFFC      DBRA D0,AGAIN
 13                      END

 ****** TOTAL ERRORS   0--   0

 SYMBOL TABLE

 AGAIN      001000
```

図 7・5 ブロック転送のプログラム例

7・3 整数算術演算命令

表7・4に示すように，加算，減算，乗算，除算，比較などを二つのオペランドを用いて行うことができます．そして，クリアやテストなどの単一オペランドでも演算が行えます．

表 7・4 整数算術演算命令

命 令	オペランド・サイズ	機 能	内 容
ADD	8,16,32 16,32	Dn+(EA)→Dn (EA)+Dn→EA (EA)+#xxx→EA An+(EA)→An	2進加算
ADDX	8,16,32 16,32	Dx+D6+X→Dx Ax@-Ay@-+X→Ax@	拡張加算
CLR	8,16,32	0→EA	クリア
CMP	8,16,32 16,32	Dn-(EA) (EA)-#xxx Ax@+-Ay@+ An-(EA)	比 較
DIVS	32÷16	Dn/(EA)→Dn	符号付き除算
DIVU	32÷16	Dn/(EA)→Dn	符号なし除算
EXT	8→16 16→32	(Dn)8→Dn16 (Dn)16→Dn32	符号拡張
MULS	16*16→32	Dn*(EA)→Dn	符号付き乗算
MULU	16*16→32	Dn*(EA)→Dn	符号なし乗算
NEG	8,16,32	0-(EA)→EA	補 数
NEGX	8,16,32	0-(EA)-X→EA	拡張付き補数
SUB	8,16,32 16,32	Dn-(EA)→Dn (EA)-Dn→EA (EA)-#xxx→EA An-(EA)→An	2進減算
SUBX	8,16,32	Dx-Dy-X→Dx Ax@--Ay@--X→Ax@	拡張付き減算
TAS	8	(EA)-0,1→EA〔7〕	オペランド・テストとセット
TST	8,16,32	(EA)-0	オペランド・テスト

〔 〕 ビット番号

ADD（アト・バイナリ）は，ソース・オペランドをディスティネーション・オペランドに加算し，結果をディスティネーションに収納します．オペレーション・サイズはバイト，ワード，ロング・ワードがあり，各データ・オペランドに対して演算を行います．加算命令には，ADDA（アドレス加算），ADDI（イミディエート加算），ADDQ（クイック加算）などのバリエーションがあります．

ADDX（アド・エクステンド）は，二つの方法によるアドレッシングによってソース・オペランドと拡張ビット（X）が行先オペランドに加算されてディスティネーションに収納されます．多倍長数，つまり32ビット以上の長さを持つデータの加算に便利です．

減算は，SUB（減算），SUBA（アドレス減算），SUBI（イミディエート減算），SUBQ（クイック減算）などのバリエーションを持ち，オペレーション・サイズは，バイト，ワード，ロング・ワードを持ちます．

SUB（サブトラクト・バイナリ）は，ソース・オペランドをディスティネーション・オペランドから減算し，結果をディスティネーションに収納します．

SUBX（サブトラクト・ウイズ・エクステンド）は，二つの方法によるアドレッシングによって，ソース・オペランドと拡張ビット（X）をディスティネーション・オペランドから減算し，結果をディスティネーションに収納します．多倍長数，つまり32ビット以上の減算に便利です．

CMP（コンペア）は，ソース・オペランドとデータ・レジスタの内容を比較して，コンディション・コードをセットします．オペレーション・サイズは，バイト，ワード，ロング・ワードがあり，CMPA（アドレス比較），CMPI（イミディエート比較），CMPM（メモリ比較）があります．

MULS（サイン・マルチプイ），**MULL**（アンサイン・マルチプイ）は，2個の符号付きおよび2個の符号なしの16ビット整数のオペランドを乗算し，32ビッの積をデータ・レジスタに入れます．また，レジスタのオペランドは下位ワードしか使用しません．

DIVS（サイン・デバイド），**DIVU**（アンサイン・デバイド）は，ディスティネーションのオペランドであるロング・ワード（32ビット）の被除数を，ソース・オペランドであるワード（16ビット）の除数で除算し，その結果を16ビット商

は上位ワードへ，16 ビット余りを下位ワードで示し，ディスティネーションに収納されます．

EXT（サイン・エクステンド）は，データ・レジスタの最上位ビットである符号ビットを，バイトからワードへ，またはワードからロング・ワードへ拡張します．

TAS（テスト・アンド・セット）は，指定されたバイト・オペランドをテストし，その結果をコンディション・コードのN（負フラグ）とZ（ゼロ・フラグ）にセットし，オペランドの最上位ビットを“1”にします．また，有効な利用方法としてマルチプロセッサ・システムやマルチタスクがあります．

TST（テスト）は，オペランドに何の影響も与えることなく，ディスティネーションをゼロと比較します．そして，テスト結果によってコンディション・コードをセットします．

CLR（クリア）は，指定されたディスティネーションをバイト，ワード，ロング・ワードの全ビットをゼロにクリアします．

ADD.W D0,D1	データ・レジスタD0の下位ワードとD1の下位ワードを加算する
SUB.B #3,(A0)	アドレス・レジスタA0の指示するアドレスの内容から“3”を引く
CMP.L $1000,D0	$1000番地のロング・ワードをデータ・レジスタD0と比較し，コンディション・コードをセットする
BCLR #3,$40(A0)	アドレス・レジスタA0の内容に$40を加えたアドレスのバイトのビット3をクリアする

図 7・6　整数算術演算命令の例

ADD：(ディスティネーション)＋(ソース)→ディスティネーション
アセンブラ：ADD <ea>,Dn
ADD Dn, <ea>

フォーマット

15	14	13	12	11 - 9	8 - 6	5 - 0
1	1	0	1	レジスタ	OPモード	実効アドレス

ADDX：(ディスティネーション)＋(ソース)＋X→ディスティネーション
アセンブラ：ADDX Dy,Dx
ADDX-(Ay),-(Ax)

フォーマット

15	14	13	12	11 - 9	8	7 - 6	5 4	3	2 - 0
1	1	0	1	Rx レジスタ	1	サイズ	00	R/M	Ry レジスタ
				①		②		③	④

① ディスティネーション・レジスタ番号

② オペレーション・サイズ
- 00：バイト・オペレーション
- 01：ワード・オペレーション
- 10：ロング・ワード・オペレーション

③ オペランドのアドレッシング・モード
- 0：データ・レジスタからデータ・レジスタ
- 1：メモリからメモリ

④ ソース・レジスタ番号

DIVU：(ディスティネーション)/(ソース)→ディスティネーション
アセンブラ：DIVU <ea>,Dn

フォーマット

15	14	13	12	11 - 9	8	7	6	5 - 0
				レジスタ	0	1	1	実効アドレス

DIVS：(ディスティネーション)/(ソース)→ディスティネーション
アセンブラ：DIVS <ea>,Dn

フォーマット

15	14	13	12	11 - 9	8	7	6	5 - 0
1	0	0	0	レジスタ	1	1	1	実効アドレス

MULU：(ディスティネーション)＊(ソース)→ディスティネーション
アセンブラ：MULU <ea>,Dn

フォーマット

15	14	13	12	11 - 9	8	7	6	5 - 0
1	1	0	0	レジスタ	0	1	1	実効アドレス

MULS：(ディスティネーション)＊(ソース)→ディスティネーション
アセンブラ：MULS <ea>,Dn

フォーマット

15	14	13	12	11 - 9	8	7	6	5 - 0
1	1	0	0	レジスタ	1	1	1	実効アドレス

CMP：(ディスティネーション)－(ソース)
アセンブラ：CMP <ea>,Dn

フォーマット

15	14	13	12	11 - 9	8 - 6	5 - 0
1	0	1	1	レジスタ	OPモード	実効アドレス

図 7・7 整数算術演算命令

SUB：(ディスティネーション)－(ソース)→ディスティネーション
アセンブラ：SUB〈ea〉,Dn
　　　　　　BUB Dn,〈ea〉

フォーマット

15	14	13	12	11–9	8–6	5–0
1	0	0	1	レジスタ	OP モード	実効アドレス

SUBX：(ディスティネーション)－(ソース)－X→ディスティネーション
アセンブラ：SUBX Dy,Dx
　　　　　　SUBX-(Ay),-(Ax)

フォーマット

15	14	13	12	11–9	8	7–6	5	4	3	2–0
1	0	0	1	Rx レジスタ ①	1	サイズ ②	0	0	R/M ③	Ry レジスタ ④

① ディスティネーション・レジスタ番号
② オペレーション・サイズ　00：バイト
　　　　　　　　　　　　　01：ワード
　　　　　　　　　　　　　10：ロング・ワード
③ オペランドのアドレッシング・モード
④ ソース・レジスタ番号

NEG：0－(ディスティネーション)→ディスティネーション
アセンブラ：NEG〈ea〉

フォーマット

15	14	13	12	11	10	9	8	7–6	5–0
0	1	0	0	0	1	0	0	サイズ	実効アドレス

NEGX：0－(ディスティネーション)－X→ディスティネーション
アセンブラ：NEGX〈ea〉

フォーマット

15	14	13	12	11	10	9	8	7–6	5–0
0	1	0	0	0	0	0	0	サイズ	実効アドレス

EXT：(ディスティネーション) 符号拡張→ディスティネーション
アセンブラ：EXT Dn

フォーマット

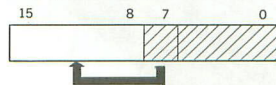
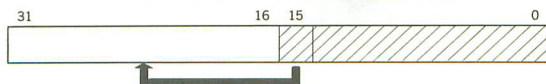
15	14	13	12	11	10	9	8–6	5	4	3	2–0
0	1	0	0	1	0	0	OP モード	0	0	0	レジスタ

EXT 命令

図 7・8 (1)　整数算術演算命令

TAS：(ディスティネーション) テスト→CC
　　　　　　　　　　　　　　　　　1→ディスティネーションの #7
アセンブラ：TAS 〈ea〉

TST：(ディスティネーション) テスト→CC
アセンブラ：TST 〈ea〉

	15	14	13	12	11	10	9	8	7 6	5 … 0
フォーマット	0	1	0	0	1	0	1	0	サイズ	実効アドレス

CLR：0→ディスティネーション
アセンブラ：CLR 〈ea〉

	15	14	13	12	11	10	9	8	7 6	5 … 0
フォーマット	0	1	0	0	0	0	1	0	サイズ	実効アドレス

図 7・8 (2)　整数算術演算命令

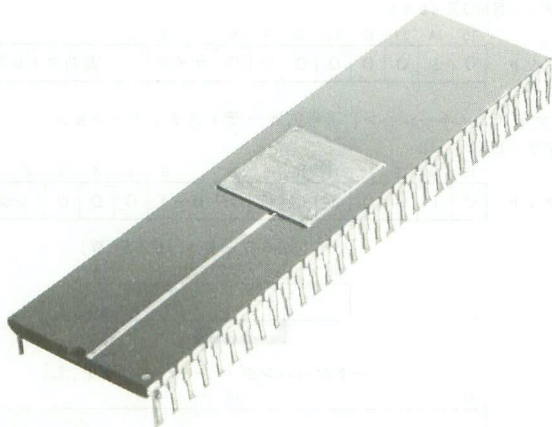
MC 68000 外観

7・4 論理操作命令

表7·5に示す論理操作命令は，バイト，ワード，ロング・ワードのいずれのオペランドにも用いることができます．このほかにANDI（アドレス・イミディエート），DRI（オア・イミディエート），EORI（エクスクルーシブ・オア・イミディエート）などがあります．

AND（アンド）命令は，ソース・オペランドとディスティネーション・オペランドとANDをとり，ディスティネーションに収納します．

OR（オア）命令は，ソース・オペランドとディスティネーション・オペランドとインクルーシブORをとり，ディスティネーションに収納します．

EOR（エクスクルーシブ・オア）命令は，ソース・オペランドとディスティネーション・オペランドとEORをとり，ディスティネーションに収納します．

NOT（ノット）命令は，ディスティネーション・オペランドの“1”の補数をとり，ディスティネーションに収納します．

表 7・5 論理操作命令

命 令	オペランド・サイズ	機 能	内 容
AND	8,16,32	Dn∧(EA)→Dn (EA)∧Dn→EA (EA)∧#xxx→EA	論理AND
OR	8,16,32	Dn∨(EA)→Dn (EA)∨Dn→EA (EA)∨#xxx→EA	論理インクルーシブOR
EOR	8,16,32	(EA)⊕Dy→EA (EA)⊕#xxx→EA	論理EOR
NOT	8,16,32	~(EA)→EA	論理否定

~：インバート

AND：(ディスティネーション)∧(ソース)→ディスティネーション

アセンブラ：AND〈ea〉,Dn
　　　　　　AND Dn,〈ea〉

フォーマット

15	14	13	12	11〜9	8〜6	5〜0
1	1	0	0	レジスタ	OP モード	実効アドレス
				①	②	③

①レジスタ番号

② 000, 100：バイト　001, 101：ワード　010, 110：ロング・ワード

③ ソース・オペランド時
　　データ・アドレッシング・モードのみ可
　ディスティネーション・オペランド時
　　可変とメモリのアドレッシング・モードのみ可

OR：(ディスティネーション)∨(ソース)→ディスティネーション

アセンブラ：OR〈ea〉,Dn
　　　　　　OR Dn,〈ea〉

フォーマット

15	14	13	12	11〜9	8〜6	5〜0
1	0	0	0	レジスタ	OP モード	実効アドレス
				①	②	③

①8データ・レジスタ番号

② 000, 100：バイト　001, 101：ワード　010, 110：ロング・ワード

③ ソース・オペランド時
　　データ・アドレッシング・モードのみ可
　ディスティネーション・オペランド時
　　可変とメモリ・アドレッシングのみ可

EOR：(ディスティネーション)⊕(ソース)→ディスティネーション

アセンブラ：EOR Dn,〈ea〉

フォーマット

15	14	13	12	11〜9	8〜6	5〜0
1	0	1	1	レジスタ	OP モード	実効アドレス
				①	②	③

① データ・レジスタ番号

② 100：バイト
　101：ワード
　110：ロング・ワード

③データと可変のアドレッシング・モードのみ可

NOT：～(ディスティネーション)→ディスティネーション

アセンブラ：NOT〈ea〉

フォーマット

15	14	13	12	11	10	9	8	7〜6	5〜0
0	1	0	0	0	1	1	0	サイズ	実効アドレス
									①

① データと可変のアドレッシング・モードのみ可

図 7・9　論理操作命令

7•5 シフトとローティット命令

シフト命令には，LSL，LSR の二つの符号なし論理シフト命令と，二つのASL，ASR の符号付き算術命令の4種類があり，ローティット命令には，ROR，ROL の二つの拡張なしローティット命令と，二つのROXL，ROXR の拡張付きローティット命令の4種類があります．またオペランドは，左右方向シフトと左右方向のローテートが可能です．表7·6 に示すように，各シフト命令の機能を有効に利用することにより，乗算命令や除算命令より効率の良い演算の実行が可能です．

ASL，ASR（アリソメティック・シフト・レフトとライト）は，ディスティネーション・オペランドのビットを指定方向に算術シフトします．シフトによって送り出されたビットは，キャリー・ビット（C）と拡張ビット（x）にそれぞれ入

表 7·6 シフトおよびローティット命令

命 令	オペランド・サイズ	機 能	内 容
ASL	8,16,32	X/C ← [←] ← 0	符号付き算術シフト
ASR	8,16,32	[→] → X/C	
LSL	8,16,32	X/C ← [←] ← 0	符号なし論理シフト
LSR	8,16,32	0 → [→] → X/C	
ROL	8,16,32	C ← [←]	拡張なしローテイト
ROR	8,16,32	[→] → C	
ROXL	8,16,32	C ← [←] ← X	拡張付きローテイト
ROXR	8,16,32	X → [→] → C	

ります．ASL 命令では，最下位ビットに 0 が入り，ASR 命令では符号ビットが最上位ビットに再度入れられます．

LSL，LSR（ロジカル・シフト・レフトとライト）は，ディスティネーション・オペランドのビットを指定方向に論理シフトします．シフトによって送り出されたビットは，キャリー・ビット（C）と拡張ビット（x）にそれぞれ入ります．LSL 命令では，最下位ビットに 0 が入り，LSR 命令では最上位ビットに 0 が入ります．

ROL，ROR（ローティット・ウイズアウト・エセステンド・レフトとライト）は，ディスティネーション・オペランドのビットを指定方向にローテートさせます．なお，拡張ビットはローテートされません．ROL 命令では，最上位から送出されたビットは，キャリー・ビットと最下位ビットの両方に入り，ROR 命令では最下位から送出されたビットはキャリー・ビットと最上位ビットの両方に入ります．

ROXL，ROXR（ローティット・ウイズ・エクステンド・レフトとライト）は，ディスティネーション・オペランドのビットを指定方向にローテートさせます．また，拡張ビットがローティットに含まれます．ROXL は，最上位ビットから送り出されたビットはキャリーと拡張の両ビットに入り，拡張ビットの前の値が最下位ビットに送り込まれます．ROXR は，最下位ビットから送り出されたビットはキャリーと拡張の両ビットに入り，拡張ビットの前の値が最上位ビットに送り込まれます．

例

LSL.W #3,D0	データ・レジスタ D0 の下位ワードを左に 3 ビット，ロジカル・シフトする．
ASR #2,(A2)	アドレスレジスタ A2 の指定するアドレスのワードを 2 ビット右にアリスメティック・シフトする．
ROXL.B D2,D1	データレジスタ D1 の下位バイトを左に，D2 の内容に従いローテション・シフトする．

ASL
ASR：（ディスティネーション）〈カウント〉によるシフト→ディスティネーション

アセンブラ：ASd　Dx，Dy
　　　　　　ASd　＃〈データ〉，Dy
　　　　　　ASd〈ea〉

フォーマット
（レジスタ・シフト）

15	14	13	12	11　　9	8	7　6	5	4　3	2　　0
1	1	1	0	カウント/レジスタ	dr	サイズ	i/r	00	レジスタ

（メモリ・シフト）

15	14	13	12	11	10	9	8	7	6	5　　　　0
1	1	1	0	0	0	0	dr	1	1	実効アドレス
							①			②

① シフト方向 ⎡0：右シフト
　　　　　　 ⎣1：左シフト

② メモリと可変アドレッシング・モードのみ可

LSR
LSL：（ディスティネーション）〈カウント〉によるシフト→ディスティネーション

アセンブラ：LSd　Dx，Dy
　　　　　　LSd　＃〈データ〉，Dy
　　　　　　LSd〈ea〉

フォーマット
（レジスタ・シフト）

15	14	13	12	11　　9	8	7　6	5	4	3	2　0
1	1	1	0	カウント/レジスタ	dr	サイズ	i/r	0	1	レジスタ
						①	②			③

① ⎡00：バイト
　 ⎢01：ワード
　 ⎣10：ロング・ワード

② ⎡0：シフト回数をイミディエート・データで指定
　 ⎣1：シフト回数をデータ・レジスタで指定

③ シフトされるデータ・レジスタの番号

15		13	12		10	9	8	7	6	5　　　　0
1	1	1	0	0	0	1	dr	1	1	実効アドレス
							①			②

① シフト方向 ⎡0：右シフト
　　　　　　 ⎣1：左シフト

② メモリと可変アドレッシング・モードのみ可

図 7・10（1）

ROL
ROR：（ディスティネーション）〈カウント〉によるローテート → ディスティネーション

アセンブラ：ROd　Dx，Dy
　　　　　　ROd　＃〈データ〉，Dy
　　　　　　ROd　〈ea〉

フォーマット（レジスタ・ローテート）

15		13	12	11　9	8	7　6	5	4	3	2　0
1	1	1	0	カウント/レジスタ	dr	サイズ	i/r	1	1	レジスタ
							①			②

① 0：ローテート回数をイミィディエート・データで指示
　 1：ローテート回数をデータ・レジスタで指示

② ローテートされるデータ・レジスタの番号

（メモリ・ローテート）

15		13	12	11	10	9	8	7	6	5　0
1	1	1	0	0	1	1	dr	1	1	実効アドレス
							①			②

① 0：右ローテート
　 1：左ローテート

② メモリと可変のアドレッシングのみ可

ROXL
ROXR：（ディスティネーション）〈カウント〉によるローテート → ディスティネーション

アセンブラ：ROXd　Dx，Dy
　　　　　　ROXd　＃〈データ〉，Dy
　　　　　　ROXd　〈ea〉

フォーマット（レジスタ・ローテート）

15		13	12	11　9	8	7　6	5	4	3	2　0
1	1	1	0	カウント/レジスタ	dr	サイズ	i/r	1	0	レジスタ
					①		②			③

① 0：右ローテート
　 1：左ローテート

② 0：ローテート回数をイミィディエート・データで指定
　 1：ローテート回数をデータ・レジスタで指定

③ ローテートされるデータ・レジスタ番号

（メモリ・ローテート）

15		13	12	11	10	9	8	7	6	5　0
1	1	1	0	0	1	0	dr	1	1	実効アドレス
							①			②

① 0：右ローテイト
　 1：左ローテイト

② メモリと可変のアドレッシング・モードのみ可

図 7・10（2）

7・6 BCD演算命令

BCD演算命令には，ABCD（拡張10進加算）命令，SBCD（拡張10進減算）命令，NBCD（拡張10進ネゲート）命令の三つがあり，2進化10進数（BCD）をバイト単位で扱う命令です．この三つの演算命令は，常に拡張フラグ（x）が含まれますので，プログラムするうえでXフラグの配慮が必要です．

ABCD（アド・デシマル・エクステンド）は，ソース・オペランドを拡張ビット（x）とともにディスティネーション・オペランドに加えて，ディスティネーションに収納します．

SBCD（サブトラクト・デシマル・エクステンド）は，ソース・オペランドと拡張ビット（x）をディスティネーション・オペランドからBCD演算を用いて減算して，ディスティネーションに収納します．

NBCD（ネゲート・デシマル・エクステンド）は，ゼロからディスティネーション・オペランドと拡張ビットを減算して，ディスティネーションに収納します．オペレーションは10進演算を用います．

表 7・7　BCD演算命令

命　令	オペランド・サイズ	機　　能	内　　容
ABCD	8	Dx10+Dy10+X→Dx Ax@-10+Ay@-10+X→Ax@	拡張付き10進加算
SBCD	8	Dx10-Dy10-X→Dx Ax@-10-Ay@-10-X→Ax@	拡張10進減算
NBCD	8	0-(EA)10-X→EA	拡張付き10進補数

ABCD：(ディスティネーション)$_{10}$＋(ソース)$_{10}$＋X → ディスティネーション

アセンブラ：ABCD Dy，Dx

ABCD -(Ay)，-(Ax)

フォーマット

15	14	13	12	11–9	8	7–4	3	2–0
1	1	0	0	Rx レジスタ	1	0 0 0 0	R/M	Ry レジスタ
					①		②	③

① ディスティネーションのレジスタ番号

② オペランドのアドレッシング・モード指定

0：データ・レジスタからデータ・レジスタへのオペレーション

1：メモリからメモリへのオペレーション

③ ソース・レジスタ番号

SBCD：(ディスティネーション)$_{10}$－(ソース)$_{10}$－X → ディスティネーション

アセンブラ：SBCD Dy，Dx

SBCD -(Ay)，-(Ax)

フォーマット

15	14–12	11–9	8	7–4	3	2–0
1	0 0 0	Rx レジスタ	1	0 0 0 0	R/M	Ry レジスタ
		①			②	③

① ディスティネーションのレジスタ番号

② オペランドのアドレッシング・モードを指定

0：データ・レジスタからデータ・レジスタ

1：メモリからメモリ

③ ソース・レジスタ番号

NBCD：0－(ディスティネーション)$_{10}$－X → ディスティネーション

アセンブラ：NBCD 〈ea〉

フォーマット

15	14	13	12	11	10–6	5–0
0	1	0	0	1	0 0 0 0 0	実効アドレス
						①

① データと可変のアドレッシング・モードのみ可

図 7・11 BCD 演算命令

7・7 ビット操作命令

ビット操作命令には，表7・8に示すようにBTST（ビット・テスト），BSET（ビット・テストとセット），BCLR（ビット・テストとクリア），BCHG（ビット・テストと変更）の4種類の命令があります．これらはいずれもバイトかロング・ワードのオペランドを用いて，ビット・テストの結果に応じてビット操作を行います．

BTST（テスト・ビット）は，ディスティネーションのうちのソース・オペランドで示されるビットの状態を調べ，そのビットの状態によりコンディション・コードのゼロ・フラグ（Z）をセットします．

BSET（テスト・ビット・アンド・セット）は，BTSTと同じ方法でビット・テストを行い，テストの後にテスト・ビットがディスティネーションによってセットされます．

BCLR（テスト・ビット・アンド・クリア）は，BTSTと同じ方法でビット・テストを行い，テストの後にテスト・ビットはディスティネーションによってクリアされます．

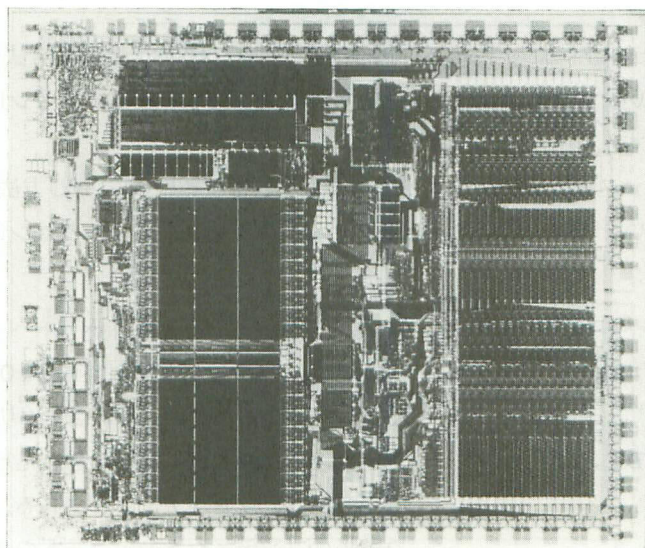
BCHG（テスト・ビット・アンド・チェンジ）は，BTSTと同じ方法でビット・テストを行い，テストの後にテスト・ビットはディスティネーションによって反転されます．

例

BCLR #2,$40(A3)	アドレス・レジスタA3の内容に$40を加えたアドレスのバイトのビット2をクリアする．
BCHG D1,D2	データ・レジスタD1で指示されたビット番号でD2のビットがチェックされる．チェック結果はコンディション・コードのZビットに表示され，D2の指定されたビットは反転される．

表 7・8　ビット操作命令

命　令	オペランド・サイズ	機　　能	内　　容
BTST	8，32	～(EA)のビット→Z	ビット・テスト
BSET	8，32	～(EA)のビット→Z 1→EA のビット	ビット・テストとセット
BCLR	8，32	～(EA)のビット→Z 0→EA のビット	ビット・テストとクリア
BCHG	8，32	～(EA)のビット→Z ～(EA)のビット→EA のビット	ビット・テストと変更

MC 68000 内部拡大図

BTST：～(ディスティネーションの〈ビット番号〉)→Z

アセンブラ：BTST　Dn，〈ea〉

　　　　　　BTST　＃〈データ〉，〈ea〉

フォーマット(ビット番号 ダイナミック)

15			12	11–9	8	7	6	5–0
0	0	0	0	レジスタ	1	0	0	実効アドレス
				①				②

① ビット番号を格納しているデータ・レジスタの番号

② アドレッシング・カテゴリのデータと可変のアドレッシング・モードのみ可

(ビット番号 スタティック)

15			12	11	10				6	5–0
0	0	0	0	1	0	0	0	0	0	実効アドレス ①
ビット番号										

① イミディエート・アドレッシング・モードを除いたアドレッシング・カテゴリのデータ可変のアドレッシング・モードのみ可

BSET：～(ディスティネーションの〈ビット番号〉)→Z

アセンブラ：BSET　Dn，〈ea〉

　　　　　　BSET　＃〈データ〉，〈ea〉

フォーマット(ビット番号 スタティック)

15			12	11	10		8	7	6	5–0
0	0	0	0	1	0	0	0	1	1	実効アドレス ①
ビット番号										

① データと可変のアドレッシング・モードのみ可

BCLR：～(ディスティネーションの〈ビット番号〉)→Z

アセンブラ：BCLR　Dn，〈ea〉

　　　　　　BCLR　＃〈データ〉，〈ea〉

フォーマット(ビット番号 ダイナミック)

15			12	11–9	8	7	6	5–0
0	0	0	0	レジスタ	1	1	0	実効アドレス

(ビット番号 スタティック)

15			12	11	10		8	7	6	5–0
0	0	0	0	1	0	0	0	1	0	実効アドレス
ビット番号										

BCHG：～(ディスティネーションの〈ビット番号〉)→Z

アセンブラ：BCHG　Dn，〈ea〉

　　　　　　BCHG　＃〈データ〉，〈ea〉

フォーマット(ビット番号 ダイナミック)

15			12	11–9	8	7	6	5–0
0	0	0	0	レジスタ	1	0	1	実効アドレス

(ビット番号 スタティック)

15	14	13	12	11	10	9	8	7	6	5–0
0	0	0	0	1	0	0	0	0	1	実効アドレス
0	0	0	0	0	0	0	0	ビット番号		

図 7・12

7・8 プログラム制御命令

プログラムの構成はジァンプやブランチを伴い，サブルーチンによるコール・プログラム・ルーチンなど，必ずしも順序正しくプログラムが実行されるとは限りません．このようなプログラムの制御命令には，表7・9に示すように条件付き，無条件，リターンと三つの分類ができます．条件付き命令にはBcc，DBcc，Sccの3種類があり，表7・10に示す16種のテスト・コンディションを備え，コンディ

表 7・9　プログラム制御命令

区　分	命　令	機　　　能
条件付き	Bcc DBcc Scc	条件付きブランチ（14条件）　8ビットと16ビット変位 テスト条件，デクリメント・カウンタとブランチ　16ビット変位 バイト条件セット（16条件）
無 条 件	BRA BSR JMP JSR	無条件ブランチ　8ビットと16ビット変位 サブルーチンへジャンプ　8ビットと16ビット変位 ジャンプ サブルーチンへジャンプ
リターン	RTR RTS	リターンと条件コード復元 サブルーチンに戻り

表 7・10　テスト・コンディション

ニーモニック	コンディション	エンコーディング	テ　ス　ト
T	トルー	0000	1
F	フェール	0001	0
HI	ハイ	0010	$\overline{C}\cdot\overline{Z}$
LS	ローかセーム	0011	$C+Z$
CC	キャリー・クリヤ	0100	$\overline{C}$
CS	キャリー・セット	0101	C
NE	ノット・イコール	0110	$\overline{Z}$
EQ	イコール	0111	Z
VC	オーバフロー・クリヤ	1000	$\overline{V}$
VS	オーバフロー・セット	1001	V
PL	プラス	1010	$\overline{N}$
MI	マイナス	1011	N
GE	大きいか等しい	1100	$N\cdot V+\overline{N}\cdot\overline{V}$
LT	小さい	1101	$N\cdot\overline{V}+\overline{N}\cdot V$
GT	大きい	1110	$N\cdot V\cdot\overline{Z}+\overline{N}\cdot\overline{V}\cdot\overline{Z}$
LE	小さいか等しい	1111	$Z+N\cdot\overline{V}+\overline{N}\cdot V$

ション付きブランチ命令とセット命令のコンディション・コードとテストの関係を示します．無条件命令にはBRA，BSR，JMP，JSRの4種類があり，リターン命令にはRTR，RTSの2種類があります．

Bcc命令は，指定されたテスト条件が満足すると，プログラムの実行はプログラム・カウンタ（PC）＋変位にブランチします．

DBcc命令は，コンディション，データレジスタ，ディスティネーションの三つのパラメータによる繰返しループ命令のターミネータとして働きます．

① 指定した条件のコンディションをテストし，満足されたかどうかを判定します．

② 条件が満たされているときは，ブランチせず，次の命令を実行します．

③ 条件が満たされない場合は，レジスタの値を“1”だけ減算します．

④ もしレジスタの値が“−1”ならばブランチせず，次の命令を実行します．

⑤ −1でないときは，PCの値に符号拡張された16ビット変位を加えた位置から続行されます．

Scc命令は，指定したコンディションを調べ，条件が満たされているときは，実効アドレスで指定されるバイトがすべて**1**にセットされ，条件が満たされない場合には，すべて**0**にセットされます．

BRA命令におけるプログラム実行は，プログラム・カウンタ（PC）＋変位（d）で続行されます．

BSR命令の直後の命令のアドレスはシステム・スタックに収納され，その後，プログラムの実行は変位の位置で続行されます．

JMP命令におけるプログラム実行は，命令で指定される制御アドレッシング・モードで指定されるアドレスで続行されます．

JSR命令の直後のアドレスがシステム・スタックに収納され，その後，プログラム実行は命令で指定されたアドレスで続行します．

RTR，**RTS**命令は，コンディション・コードとPCをスタックから取り出します．また，命令実行前のコンディション・コードとPCは消去されます．

B_{CC}：もしccならばPC+d→PC
アセンブラ：Bcc〈ラベル〉

フォーマット

DBcc：もしccでなければDn−1→Dn
もしDn≠1ならばPC+d→PC
アセンブラ：DBcc Dn，〈ラベル〉

フォーマット

Scc：もしccならば"1"→ディスティネーション
もしccでない"0"→ディスティネーション
アセンブラ：Scc〈ea〉

フォーマット

15	14	13	12	11–8	7	6	5–0
0	1	0	1	条件	1	1	実効アドレス

BRA：PC+d→PC
アセンブラ：BRA〈ラベル〉

フォーマット

BSR：PC→SP@−；PC−d→PC
アセンブラ：BSR〈ラベル〉

フォーマット

JMP：ディスティネーション→PC
アセンブラ：JMP〈ea〉

フォーマット

15	14	13	12	11	10	9	8	7	6	5–0
0	1	0	0	1	1	1	0	1	1	実効アドレス ①

① ジャンプ先のアドレス指定，制御アドレシング・モードのみ可

JSR：PC→SP@−；ディスティネーション→PC
アセンブラ：JSR〈ea〉

フォーマット

15	14	13	12	11	10	9	8	7	6	5–0
0	1	0	0	1	1	1	0	1	0	実効アドレス ①

① ジャンプ先のアドレス指定，制御アドレシング・モードのみ可

RTR：SP@+→CC; SP@+→PC
アセンブラ：RTR

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
フォーマット	0	1	0	0	1	1	1	0	0	1	1	1	0	1	1	1

RTS：SP@+→PC
アセンブラ：RTS

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
フォーマット	0	1	0	0	1	1	1	0	0	1	1	1	0	1	0	1

図 7・13　プログラム制御命令

```
*   SCC INSTRUCTION EXAMPLE
*
*THE SCC INSTRUCTION ALLOWS THE USER
*TO REMEMBER THE RESULT OF AN INSTRUCTION
*AND ACT UPON IT AT SOME LATER TIME
*
 ORG $2000
 CLR D0
 CLR D1
 CMP D0,D1
 SEQ D2
 ADDQ #1,D0
 CMP D0,D1
 SEQ D2
 END
```

図 7・14　SCC プログラムの例

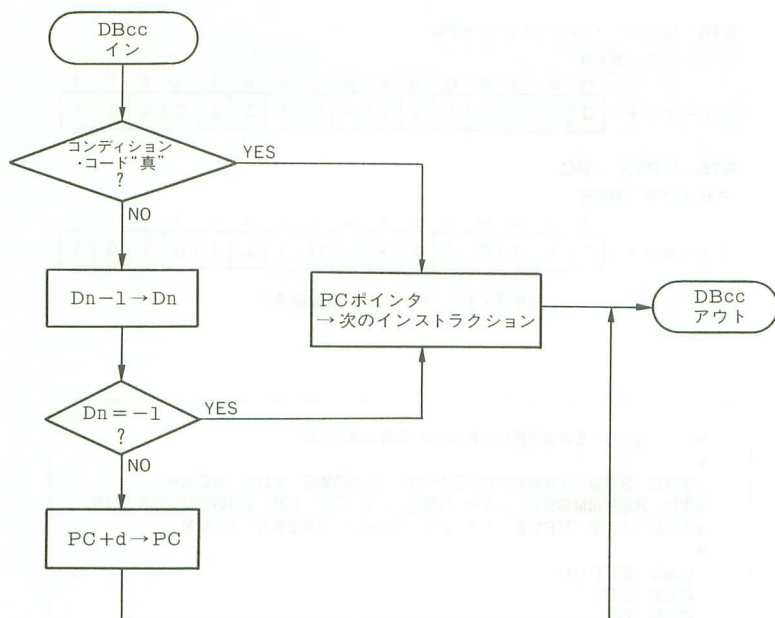



図 7・15　DBcc 命令フロー図

条件付きジャンプ	
DBEQ　DO,$1000	コンディションが"EQ"ならば次の命令を実行し，"EQ"でないならば"DO"から"1"引く．そして，引かれた結果がマイナスならば次の命令を実行する．マイナスでなければ$1000番地に飛ぶ
BNE.L　$5000	コンディションが"NE"のとき$5000番地へロング・ブランチする
BEQ.S　$5000	コンディションが"EQ"のとき$5000番地へショート・ブランチする
無条件ジャンプ	
JMP　2(A7)	A7に2を加えたアドレスへ飛ぶ
JMP.L　$1000	$1000番地へ飛ぶ．アブソリュート・ロング・ジャンプ
JSR　$5000	$5000番地のサブルーチンへ飛ぶ
BSR　$5000	$5000番地にあるサブルーチンへ飛ぶ

図 7・16　プログラム制御命令の例

7・9 リンク命令とアンリンク命令

リンク命令とアンリンク命令は，メモリ・スタックの割当てや復帰のための手順やサブルーチン用法などの命令です．そして，有効なデータ領域への使用や，作業領域の確保としてのシステム・スタック用法でもあります．また，プログラミングするうえで非常に有益な命令であり，リンク命令によってスタック・ポインタをどのように変更しても，UNLK 命令を使用することによりフレーム・ポインタやスタック・ポインタがLINK 命令前の値に自動的に復帰されることは大きな魅力といえます．

LINK（リンク）命令は，指定されたアドレス・レジスタの内容がスタックに収納され，その後，プリデクリメント（−4）されたスタック・ポインタの値がアドレス・レジスタにロードされます．そして，符号拡張されたディスプレースメントがスタック・ポインタに加えられます．

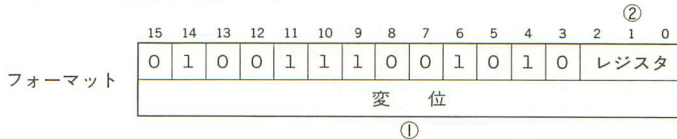
LINK：An → SP@−; SP → An; SP+d → SP
アセンブラ：LINK An，# 〈ディスプレースメント〉

① スタック・ポインタに加算する“2”の補数の整数
② リンクするためのアドレス・レジスタの番号

図 7・17

UNLK（アンリンク）命令は，指定されたアドレス・レジスタがスタック・ポインタにロードされ，アドレス・レジスタへスタックの先頭から取り出されたロング・ワードがロードされます．

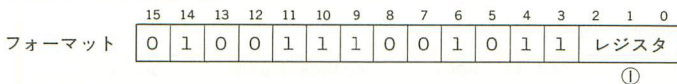
UNLK：An → SP; SP@+ → An
アセンブラ：UNLK An

① アドレス・レジスタ番号

図 7・18

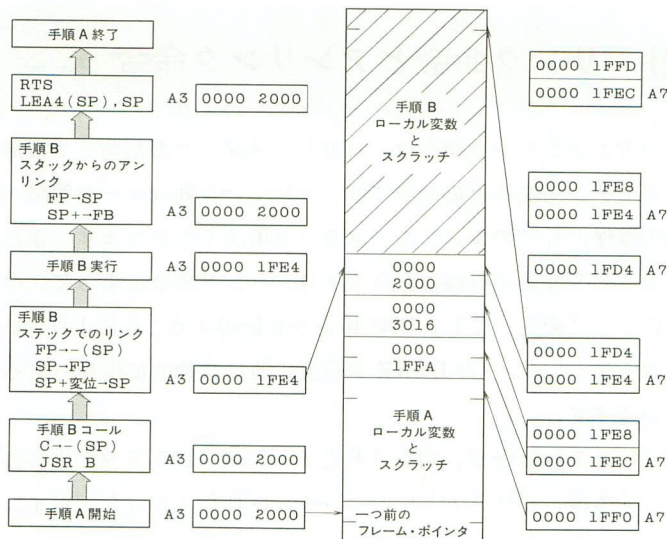
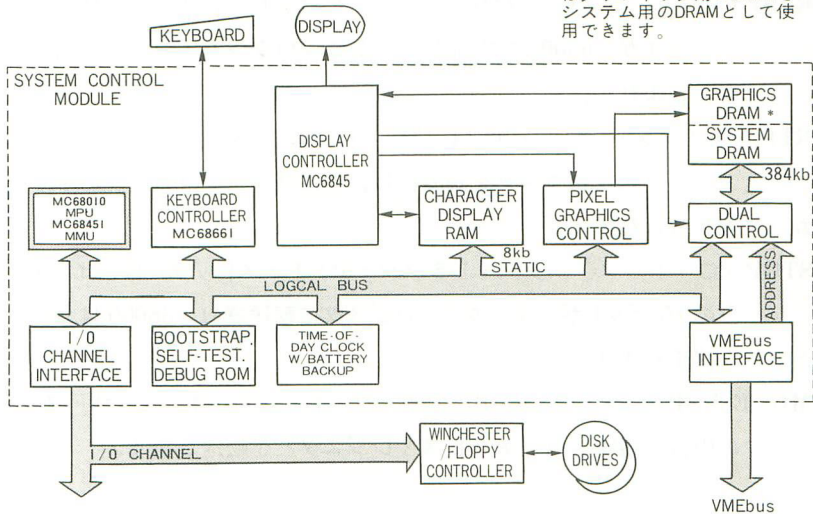



図 7・19　プログラム・フロー図

```
LINKUN     MC68000 ASM


  1          00003000      ORG $3000
  2 003000 47F82000        LEA $2000,A3
  3 003004 4FF81FF0        LEA $1FF0,SP
  4 003008 4E71          PROCA NOP ──────────── ラベル "PROCA"
  5 00300A 4E71            NOP
  6 00300C 4E71            NOP
  7 00300E 486BFFFA        PEA -6(A3)
  8 003012 4EB83022        JSR PROCB ─────────── リンク・ルーチンへジャンプ
  9 003016 4FEF0004        LEA 4(SP),SP
 10 00301A 4E71            NOP
 11 00301C 4E71            NOP
 12 00301E 4E71            NOP
 13 003020 60FE          ENDA BRA * ──────────── ラベル "ENDA"
 14 003022 4E53FFF0      PROCB LINK A3,-$10 ──── ラベル "PROCB"
 15 003026 4E71            NOP
 16 003028 4E71            NOP
 17 00302A 4E71            NOP
 18 00302C 4E5B            UNLK A3
 19 00302E 4E75            RTS ───────────────── リターン
 20                        END

 ****** TOTAL ERRORS    0--    0


 SYMBOL TABLE

 ENDA        003020 PROCA       003008 PROCB       003022
```

図 7・20　リンク，アンリンクのプログラム例

■ VME/10 システムブロック図

VME 10システム・ブロック図

特　長

1. MC 68010 16ビット仮想マシン搭載.
2. MC 68451 メモリ管理用 LSI 搭載.
3. VME bus バス仕様と I/O Channel インタフェース.
4. 34 K バイトのダイナミック RAM と 8 K バイトのスタティック RAM 実装.
5. 表示形式をソフトウェアで制御できる.
6. フロッピー・ディスク（1 M バイトのアンフォマット）.
7. ウィンチェスタ・ディスク（6 M/19 M バイトのアンフォマット）.

VME/10 マイクロコンピュータ・システム（その2）〈その3は p. 143〉

7・10 システム制御命令

システム制御命令は，表7・11に示すように特権命令，トラップ命令，ステータス・レジスタ命令と機能別に3タイプに分類できます．

RESET（リセット）は，リセット・ラインをセットし，すべての外部デバイスをリセットしますが，68000の内部レジスタはプログラム・カウンタ以外のレジスタ群は変化しません．

RTE（リターン・フロム・エクセプション）は，スタックからプログラム・カウンタとステータス・レジスタを読み出し，プログラム・カウンタの示すアドレスから処理を続行します．

STOP（ロード・ステータス・レジスタ・アンド・ストップ）は，イミディエート・データの16ビットがステータス・レジスタに転送され，68000は命令フェッチと実行を停止します．

MOVE USP（ムーブ・ユーザ・スタック・ポインタ）は，ユーザ・スタック・ポインタの内容と指定されたアドレス・レジスタとを転送します．

表7・11　システム制御命令

区　分	命　令	機　能
特権命令	RESET	外部デバイス・リセット
	RTE	例外からのリターン
	STOP	プログラム例外のストップ
	ORI to SR	ステータス・レジスタへの論理OR
	MOVE USP	ユーザ・スタック・ポインタへの転送
	ANDI to SR	ステータス・レジスタへの論理AND
	EORI to SR	ステータス・レジスタへの論理EOR
	MOVE EA to SR	新しいステータス・レジスタのロード
トラップ命令	TRAP	トラップ
	TRAPV	オーバフロー・トラップ
	CHK	レジスタ限界チェック
ステータス・レジスタ	ANDI to CCR	コンディション・コードへの論理AND
	EORI to CCR	コンディション・コードへの論理EOR
	MOVE EA to CCR	新しいコンディション・コードのロード
	ORI to CCR	コンディション・コードへの論理OR
	MOVE SR to EA	ステータス・レジスタの収納

TRAP（トラップ）は，68000 が無条件で例外処理を開始します．命令の下位4ビットで指定されるトラップ命令の例外ベクタを参照するようベクタ番号が作られます．

TRAPV（トラップ・オン・オーバフロー）は，68000 がオーバフロー状態のときに例外処理を開始します．

CHK（チェック・レジスタ・アゲンスト・ボウンダ）は，指定したデータ・レジスタの下位ワードの値とソース・オペランドで示される“2”の補数の整数と比較し，レジスタ値が**0**より小さいか，指定した整数値より大きいと，68000 は例外処理を開始します．

ステータス・レジスタ命令は，コンディション・コードに対する論理AND，論理EOR，論理OR のオペレーションや，コンディション・コードのセッティングやレジスタへの収納などの命令です．

RESET：――――
アセンブラ：RESET

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
フォーマット	0	1	0	0	1	1	1	0	0	1	1	1	0	0	0	0

RTE：SP@+→SR
　　　SR@+→PC
アセンブラ：RTE

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
フォーマット	0	1	0	0	1	1	1	0	0	1	1	1	0	0	1	1

STOP：イミディエート・データ→SR
アセンブラ：STOP　#×××

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
フォーマット	0	1	0	0	1	1	1	0	0	1	1	1	0	0	1	1
	イミディエート・データ ①															

① SR にロードするイミディエート・データを指定

図 7・21（1）

MOVE USP：USP→An
　　　　　An→USP
アセンブラ：MOVE USP,An
　　　　　　MOVE An,USP

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
フォーマット	0	1	0	0	1	1	1	0	0	1	1	0	dr	レジスタ		
													①	②		

① アドレス・レジスタをユーザ・スタック・ポインタへ転送：0
　ユーザ・スタック・ポインタをアドレス・レジスタへ転送：1
② アドレス・レジスタ番号

TRAP：PC→SSP@-
　　　SR→SSP@-
　　　(ベクタ)→PC
アセンブラ：TRAP ＃〈ベクタ〉

	15	14	13	12	11	10	9	8	7	6	5	4	3			0
フォーマット	0	1	0	0	1	1	1	0	0	1	0	0	ベクタ			
													①			

① 16個のベクタ番号の指定

TRAPV：もしVならTRAP
アセンブラ：TRAPV

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
フォーマット	0	1	0	0	1	1	1	0	0	1	1	1	0	1	1	0

CHK：もしDn〈0かDn〉(〈ea〉)ならTRAP
アセンブラ：CHK〈ea〉,Dn

	15	14	13	12	11		9	8	7	6	5					0
フォーマット	0	1	0	0	レジスタ			1	1	0	実効アドレス					
					①						②					

① データ・レジスタ番号．この内容が調べられる
② ソース・オペランドを指定し，この内容が上限値．
　データ・アドレッシング・モードのみ可

図 7・21 (2)

8. システムの構成

68000のシステム構成を，基本的なシステム構成をベースに，最小システム構成の条件や68000ファミリ・チップを用いた中位システム構成，上位システム構成へと展開し，またシステムとVERSAbus，モジュールの関係を説明します.

8・1 最小システム

68000における最小システム構成は，図8・2に示すような**四つ**の基本的機能が要求されます．これらの基本構成に基づいて構成されたシステムが，図8・1(a)と(b)です．メモリや6800 I/Oの仕様は，MC68000をどのようなスピードのものを使用するか，また，どのようなI/Oを使用するかによって分かれてきます．

例えば，メモリ関係は**MC6810**スタティックRAM，**MC6829** MMUがあり，DMAコントローラには**MC6844**（並列転送），**MC6855**（直列転送）があり，並列I/Oインタフェースには**MC6821**（PIA），直列インタフェースには**MC6850**（非同期型），**MC6852**（同期型），**MC6854**（データ・リンク型）があり，ディスプレイ・コントローラには**MC6845**（CRTコントローラ），**MC6847**（ディスプレイ・コントローラ）などがあります．

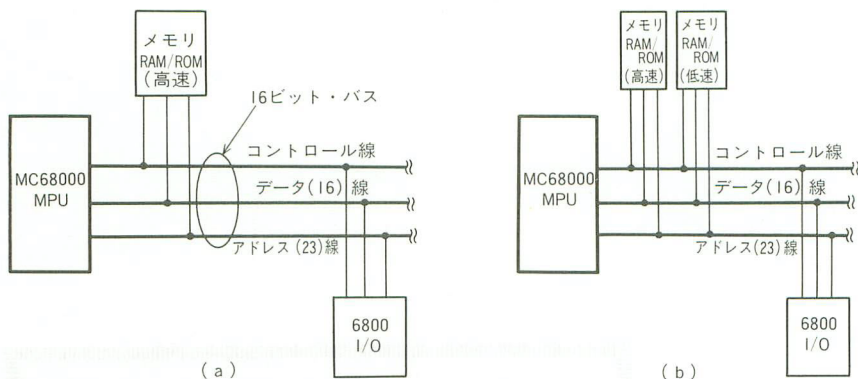



図8・1 68000最小システム

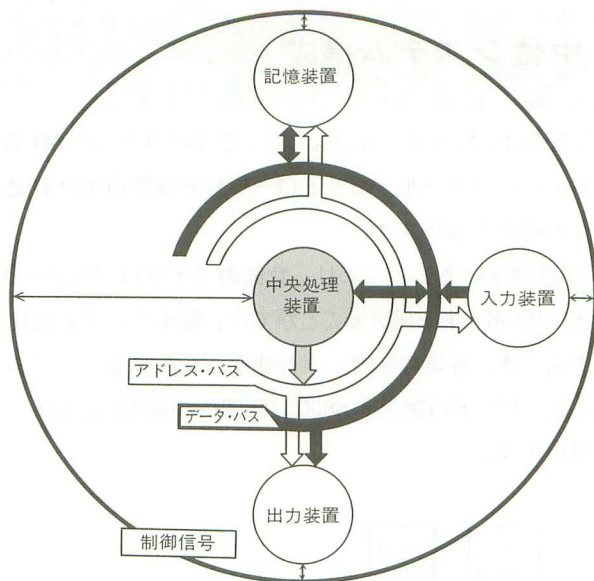



図 8・2 システムの基本構成

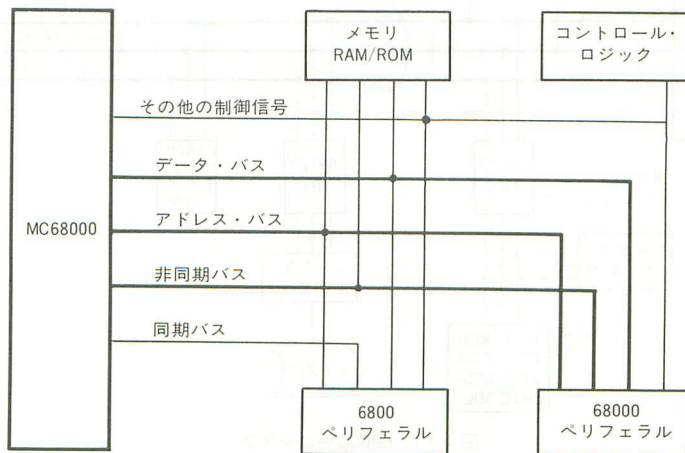



図 8・3 68000の基本システム

8·2 中位システム構成

中位システム構成は，図8·4に示すように，最小システムに**MC68120** IPC（インテリジェント・ペリフェラル・コントローラ）が数個追加されることにより，中規模システムの構成となります．

数個のIPCが追加されることにより，数個のマイクロプロセッサやメモリ（RAM，ROM）やI/Oポートを備えることができ，最小システムに比べ高効率な**分散処理**形式が構成でき，各処理速度，実行速度が数段異なってくることです．また，68000周波数スピードの選択や6800 I/O関係の選択は，最小システムと同様に選択して使用します．

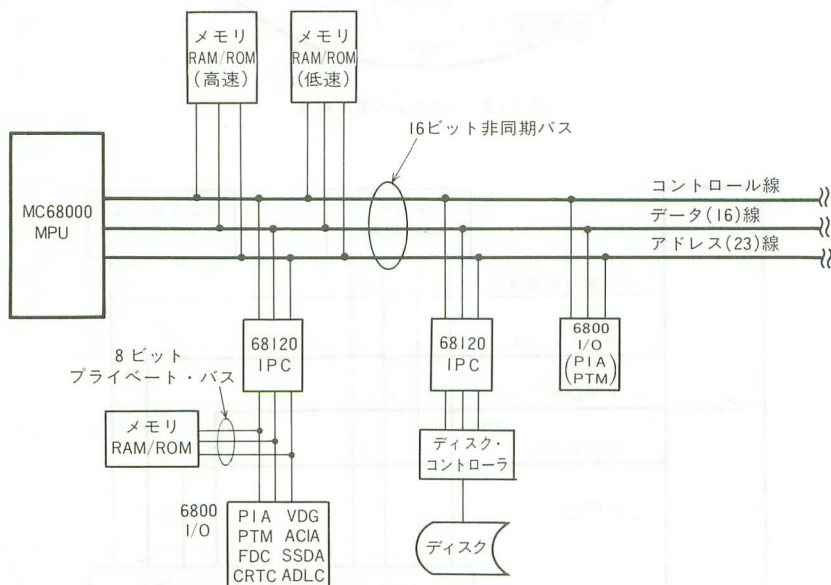



図 8·4　68000中位システム

8・3 上位システム構成

上位システム構成は，マルチチャネルを備えることによってマルチユーザ・システムを構成し，リアルタイム・バス・ステートやリモート開発ステーションを備えることができます．

図8･5に示すように，中位システム機能のほかに，**MC68450** DMAC，**MC68451** MMU，**MC68230** PITなどの機能が追加されます．メモリ管理ユニット（MMU）の追加によって，68000の16Mバイトのアドレス空間のアドレス変換とメモリ保護が強化され，しかも大容量のメモリを高効率に管理でき，オペレーティング・システムのオーバヘッドを減少させることができます．また，ダイレクト・メモリ・アクセス・コントローラ（DMAC）の追加によって，四つの完全に独立したチャネルを持つことができ，4Mバイト/秒という高速なデータ転送が実行できます．

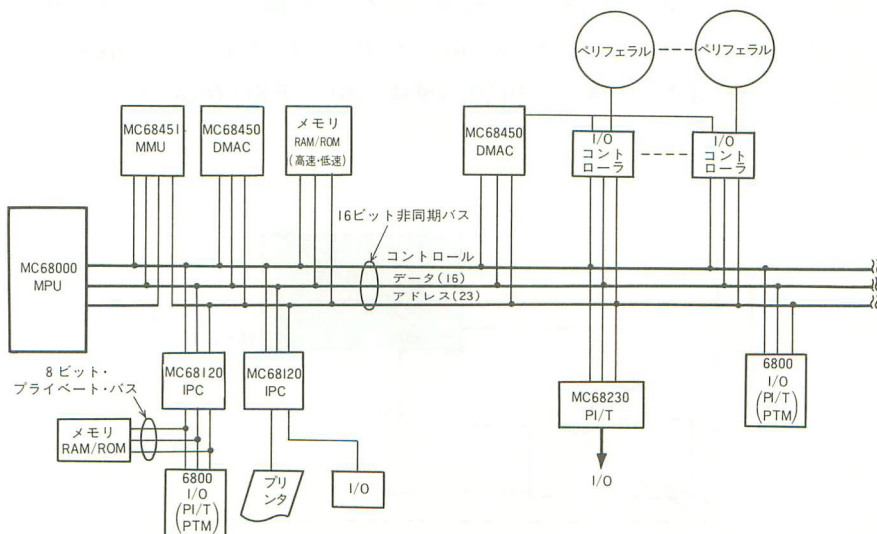



図 8・5 68000上位システム

8・4 システムとVERSAbusの関係

VERSAbusは，スピード・アプリケーション用として，MC68000用のボード，シャーシ，システム・ソフトウェアなどを応用機器製作要素としてモジュール化したファミリです．

バーサ・バスは，マルチプロセス・システムのバス・ストラクチャをとっており，8ビット，16ビット，32ビットの非同期式データと高速度のデータの転送を行います．そして，32アドレス・ラインによって，40億バイトのメモリを直接アクセスできます．またこれらのデータは，デバイス間のブロック・データ転送が**5レベル**のバスの優先スケマティックによってスムーズに行われます．そして，VERSAbusストラクチャにおけるバス・クリア・コマンドを介してダイナミックに優先度を適応させます．

図8・6は，各種のモジュールとパフォーマンスの関係および推移を示したものです．また，価格とパフォーマンスの関係も選択の重要な要素の一ついえます．

図8・8は，バーサ・バスをベースに基本システムを構成した場合，拡張サポートの場合，マルチユーザ・システムの場合のブロック図とおのおのの関係を図示したものです．また，システムの拡張は単純であり，EXORmacsのシャーシに，

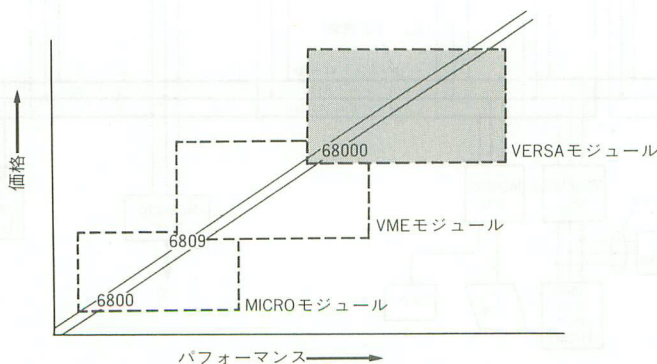



図 8・6 モジュールとパフォーマンスの関係

それぞれ適応するプラグ式のペリフェラルやモジュールを差し込むだけでよく，VERSdos オペレーティング・システムのアップデート・デバイス・テーブルと **SYSGEN** ユーティリティを用いて実行できます．

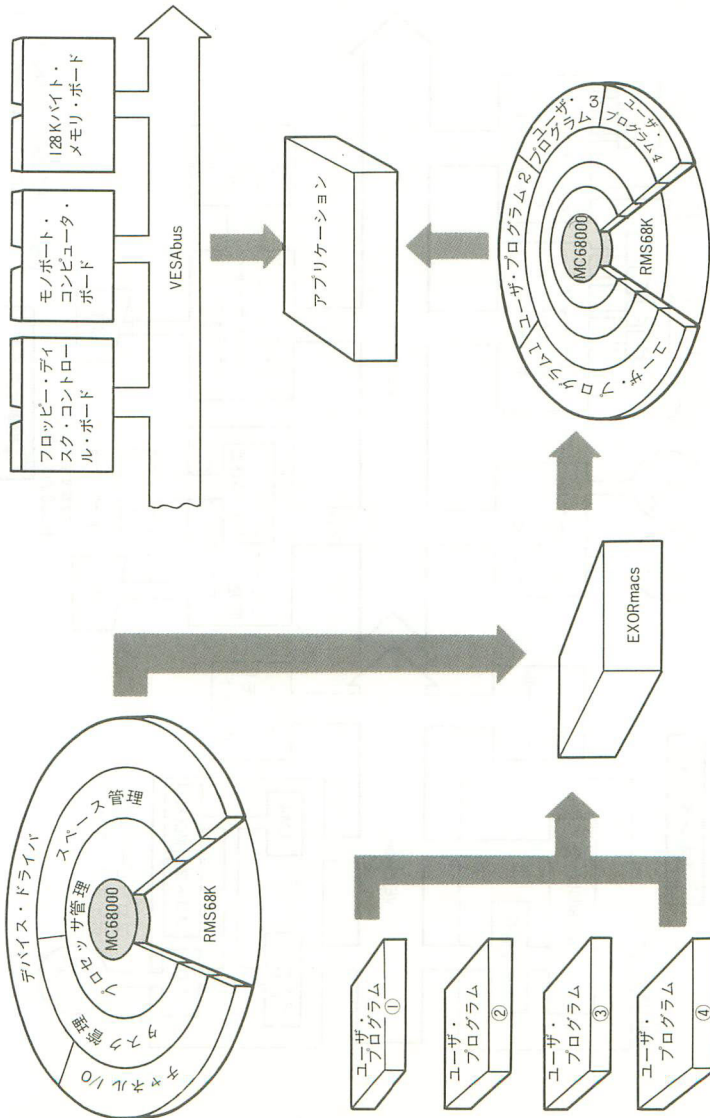



図 8・7 スピード・アプリケーション・バーサ・バス

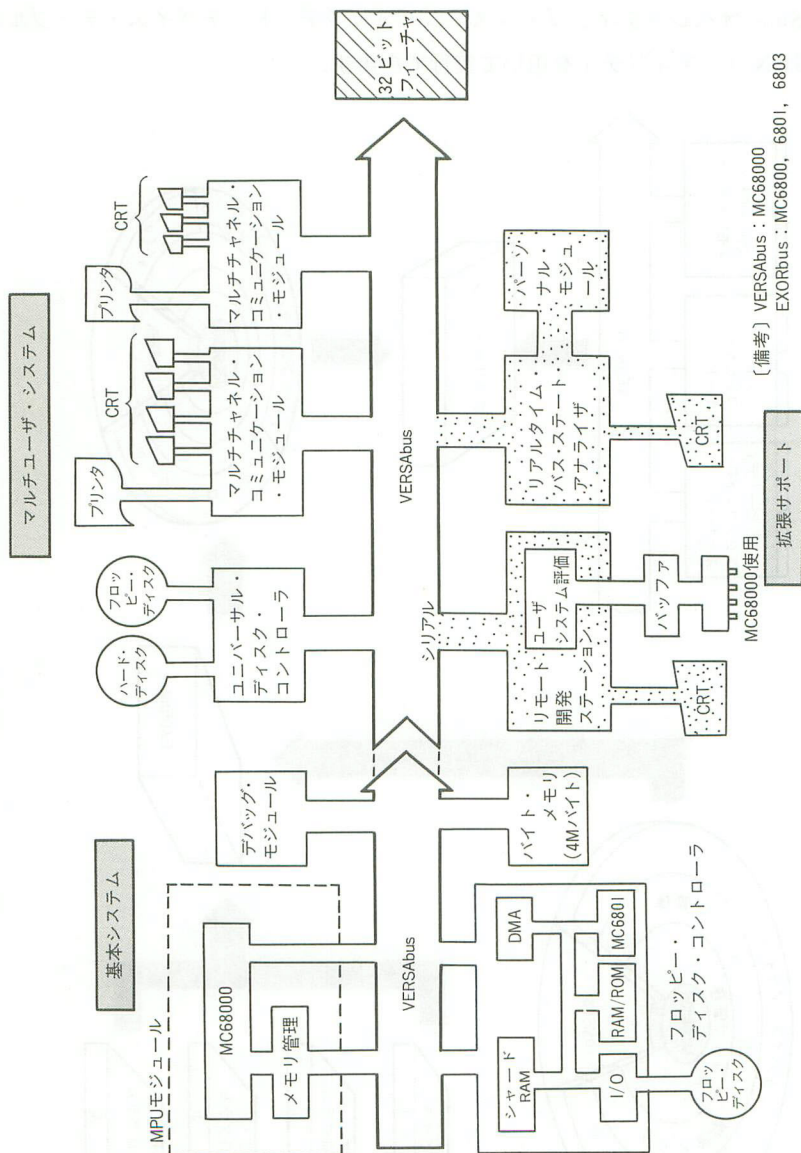
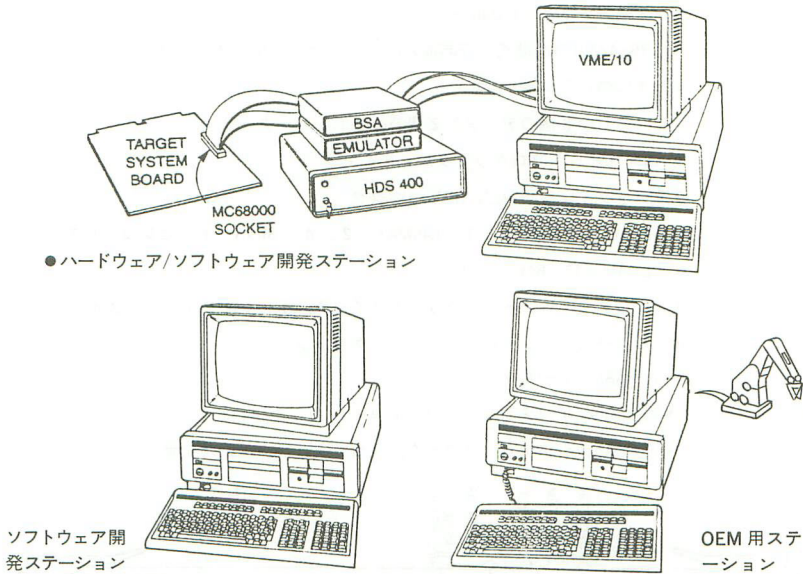
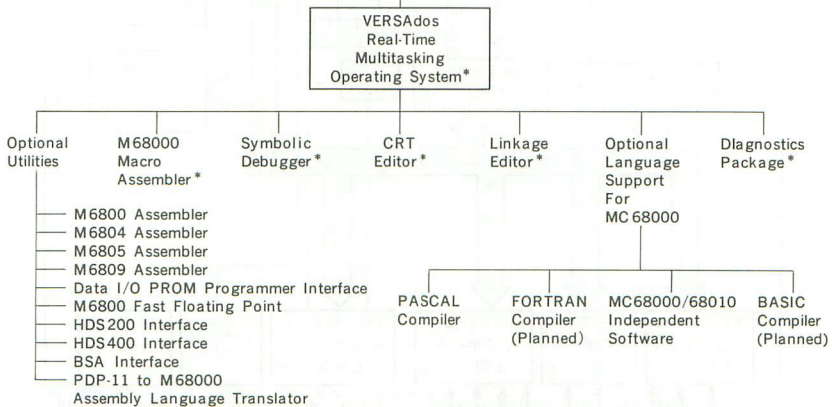



図 8・8 システムとの関係

VME/10 マイクロコンピュータ・システム (その3)

モノボード・マイクロコンピュータ

① MC68000 MPU(8MHz)

② VERSAbusインタフェース

③ 2個のシリアルI/Oポート

RS-422用の同期式，非同期式のプログラマブルとストラパブル

④ 4個のパラレルI/Oポート

おのおの8個のデータと2個のハンドシェーク・ライン

⑤ 3×16ビットプログラマブル・タイマ

⑥ バックプレーン 入力/出力 コネクタ

⑦ ROM，EPROMソケット（64KMAX，2，4，8Kバイト・ピンコンパチ）

⑧ DRAM（32～64Kバイト）

⑨ システム・テストとリセットのスイッチ，ボード・ステータス表示

⑩ システム・コントロール・ファンクション

- VERSAbusアービタ
- VERSAbusインタラプト・ハンドラ
- VERSAbusシステム・クロック，リセット，テスト，その他

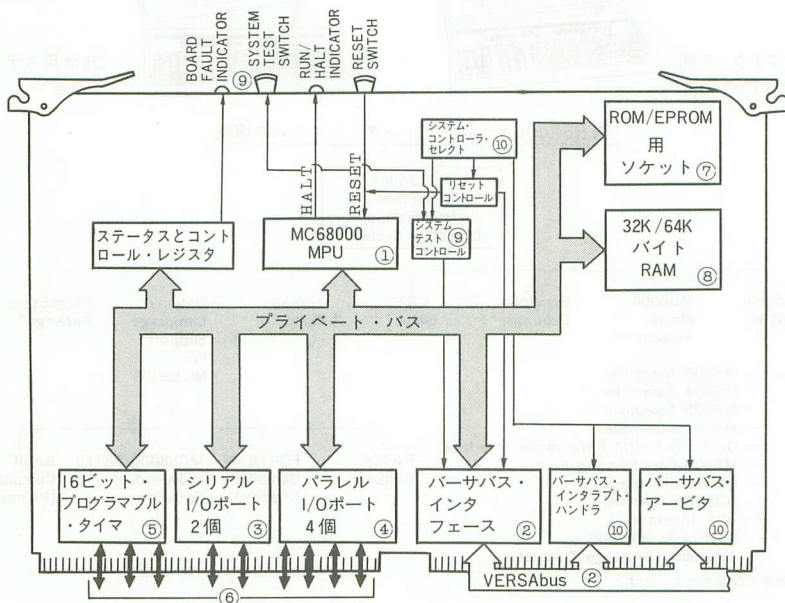



図 8・9

9. 周辺ファミリ・チップ

いままで述べたようにMC68000は，他のプロセッサに類のない卓越した性能を持つマイクロプロセッサです．しかし，その性能をシステムで生かすには，周辺のLSIのサポートがあってこそです．この章では，これらの周辺チップの開発予定を明らかにするとともに，特に重要と思われる周辺チップについては，若干の機能の説明をしました．

9・1 MC68000をサポートする周辺チップ

MC68000を中心にしたマイクロコンピュータ・システムにおいて，システムの性能や信頼性向上は，MC68000周辺チップのサポートを欠かすことができません．

モトローラ社は，図9・1に示すように，これらの周辺チップの開発，発売を予定しており，現在すでにMC68451（MMU：メモリ管理ユニット），MC68230（PI/T：並列インタフェース用ICタイマ内蔵），MC68120（IPC：インテリジェント・ペリフェラル・コントローラ）および数種の通信用LSIを入手可能にしています．また，バーチャル機構を実現し，演算命令などの改良により実行スピードが向上した16/32ビット・マイクロプロセッサMC68010もすでに発売中です．MC68010は，MC68000とピン・コンパチブルで，命令などもアップワード・コンパチブルですので，MC68000で組まれたシステムをハードウェアの変更なしに置き換えるだけで，システムのパフォーマンス・アップを図ることができます．

MC68000周辺チップは，大きく分けて三つに分類できます．それは，直接MC68000の出力する論理アドレスをシステムのグローバルの物理アドレスに変換するメモリ管理ユニットと，各種コントローラおよび各種インタフェース用チップです．

さらにMC68000ファミリは，単なる16/32ビットの新たなファミリ群として登場したわけではなく，8ビットのシステム・バスを持つMC6800ファミリに対して上位互換性を保っています．したがって，MC68000は，その同期式のバス・コントロール・ラインを用いて直接MC6800ファミリを接続し，周辺チップとして働かせることができます（図9・2参照）．

システムの性能向上の手段としては，クロック・スピードの速いMPUを用いることがあげられます．現状では，12MHzのクロック・スピードを持つMC68000L12まで入手可能ですが，アクセス・タイムの遅いメモリや遅いクロック動作の周辺チップでは，スループットの向上は見込めません．周辺チップのクロック・スピードも向上する方向にありますが，今は8MHzが主流となっており，コス

ト・パフォーマンスの点からも，システム・クロックを決める場合，周辺チップのクロック・スピードがかぎになっているのが現状です．

浮動小数点演算
コ・プロセッサ
MC68881
FPCP

バス分散
コントローラ
MC68452
BAM

ディスク・
コントローラ
MC68454
IMDC

分散処理
コントローラ
MC68120
IPC

MPU
MC68000
MC68008
MC68010
MC68020

データ・ライン
アドレス・ライン

MC68451
MMU
メモリ管理
ユニット

MC68440
DDMA
MC68450
DMAC
DMA
コントローラ

MC68230
PI/T
MC68901
MFP
M6800ファミリ
周辺チップ
汎用I/O
コントローラ

MC68652
MPCC
MC68653
PGC
MC68661
EPCI
MC68681
DUART
MC68562
DUSCC
MC68564
SIO
データ通信用
デバイス

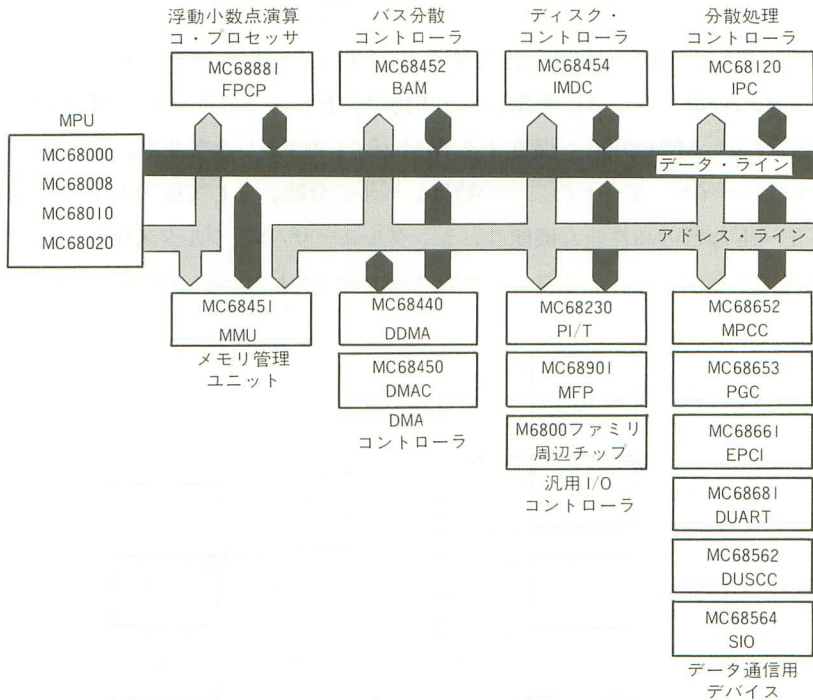



図 9・1 68000ファミリ・チップ

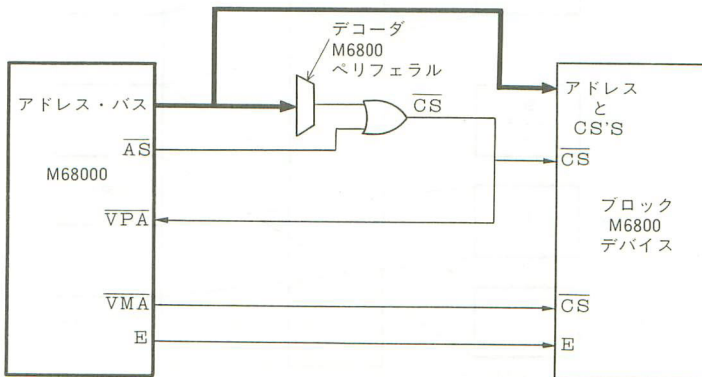



図 9・2 68000と6800の接続ペリフェラル

9・2 メモリ管理ユニット MC68451

MC68451は，MC68000が出力する16Mバイトの論理アドレス空間を，実際に存在する物理アドレスに変換し，その物理アドレスの管理をするMC68000周辺LSIです．MC68000が保有する16Mバイトのアドレス空間や，ユーザ・モード/スーパバイザ・モードの二つの特権レベルの分離，そして強力なアドレッシング・モードなどの高性能な機能は，シングルユーザ/シングルタスク，バッチ指向の処理にとどまらず，ミニコン・クラス以上のコンピュータが実現していたマル

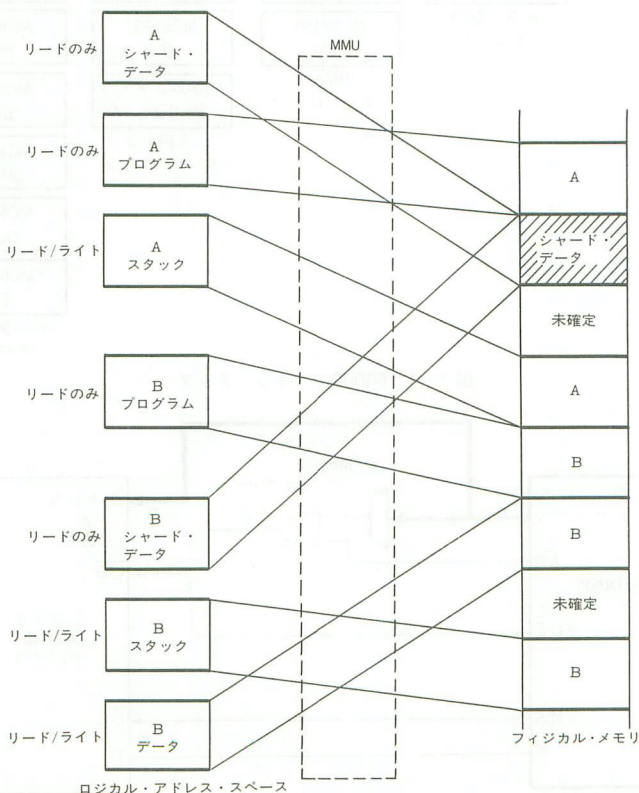



図 9・3 フィジカル・メモリにおけるマッピング・ロジカル・セグメント

チユーザ/マルチタスク，リアルタイムなどの処理を十分サポートできるものです．これらのマルチ処理を行うには，当然，タスク間/システム・プログラムとユーザ・プログラム間/ユーザ間の相互アクセスをプログラマブルで禁止し，メモリ内の割付け，メモリの管理，保護を実行するメモリ管理用のチップMC68451が必要になります．

MC68451MMU（MEMORY MANAGMENT UNIT）の主な機能は，図9·3に示すような論理アドレスの，実際に存在する物理アドレスへの変換にあります．また，MC68000と1個のMMU，そしてメモリからなる最小システムを図9·4に示し，MMU内部の機能ブロック図を図9·5に示します．

MMUは，メモリ空間がページ（**256**バイト～**16 M**バイト，2^nバイト$8 \leqq n \leqq 24$）に分割された空間を，それぞれ論理/物理アドレス空間に割り付け，変換テーブルにより相互の対応をとるページング方式を用いています．

MMUは，図9·3に示す各アドレス空間の変換に際して，前もって内部のレジスタに設定された書込み保護，ファンクション・コードの指定（スーパバイザ/ユーザの分離，プログラム/データの分離），アドレス空間の位置，大きさなどを

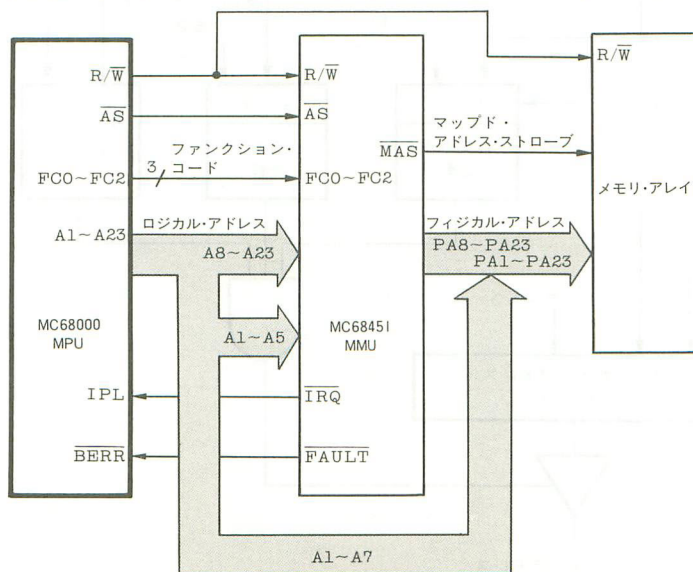



図 9・4 シングルMMUシステム・ブロック図

図9·5で示されるようなフローでチェックし，もしそれらに違反があると，割込みやフォルトなどでプロセッサに知らせます．

また，MC68451の特長として，一つのシステムに複数のMPUを共有できることがあげられます．おのおののMMUは，アドレス空間を**32**個のセグメントに分割して管理，保護する能力を持っていますが，それ以上のセグメントが必要な場合に複数のMMUを使用します．

MC68451は，前に述べたセグメント・フォルト（書込み/セグメント範囲違反など）をプロセッサに知らせる機能により，仮想記憶制御に対応します．しかし，それにはプロセッサ自体にセグメント・フォルトの要因解除後の再開始機能が含

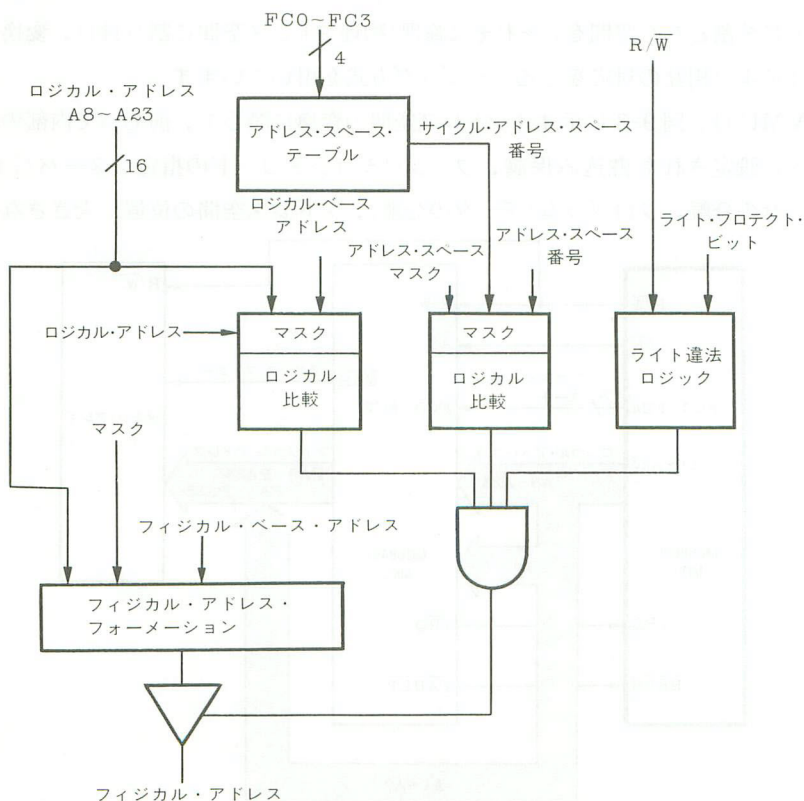



図 9·5 MMUのファンクション・ブロック図

まれていなければなりません．この機能を実現するには，MC68010（バーチャル・マシン）が持つ例外処理の強化や，フォルト発生時，プロセッサが内部状態（データ/アドレス・レジスタ，プログラム・カウンタ，ファンクション・コード，中断された命令など）を再開始可能に十分なだけスタックする機能を備えることによりなされます．

MC68000は，MC68010のような命令の再開始機能は備えていませんが，図9·6のように2台のプロセッサを用いて仮想記憶制御を実現します．図中のMPU1が通常の処理を行う主プロセッサとなり，MPU2が，MPU1をバック・アップして仮想記憶制御を実行します．MPU2は，セグメント不在のフォルトをバス・エラー入力により検出すると，MPU1への$\overline{\text{DTACK}}$（データ転送アクノリッジ）入力信号を強制的に返さないようにし，MPU1をウエート・ステート（$\overline{\text{DTACK}}$待ち）状態にします．この間MPU2がバス・マスタとなり，二次記憶機器などのハード・ディスクから主記憶上に不在であったデータを主記憶にセグメント・スワップし，MMUのセグメント・ディスクリプタ更新をします．その後$\overline{\text{DTACK}}$をMPU1に返すことにより，MPU1は中断されていた命令を開始し，通常の処理を続行します．

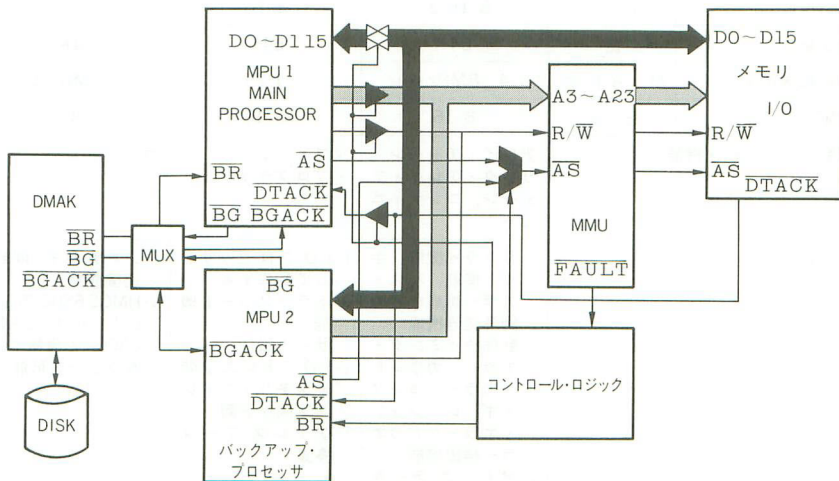



図 9·6　MC68000とMC68451におけるバーチャル・メモリ例

9・3 DMAコントローラMC68450

16ビットのマイクロコンピュータ・システム，特にMC68000をCPUとした高性能なミニコン・クラスのシステムでは，大量のデータが必要となります．また，リアル・タイム処理を実行するにあたっては，入出力機器から随時データを入力し，システムの主記憶に高速データ転送をしなければなりません．

このように高速CRT端末，ハード・ディスクおよびフロッピー・ディスクなどの大量のデータ入出力機器を持ったシステムや，メモリ内をタスク分けしてデータやプログラム域の整備を必要とするシステムでは，入出力機器と主記憶，あるいは主記憶間での大量のデータ転送が要求されます．MC68000自身も，MOVE

表9・1 DMAコントローラの比較

項目	単位	MC68450	i8089	MC6844
クロック周波数	MHz	4，6，8（予定）	5	1，1.5，2
パッケージ・ピン数	本	64	40	40
チャネル数	本	4	2	4
直接アドレス空間	バイト	16M	1M	64K
転送単位	ビット	8/16/32	8/16	8
転送語数	語	64K	64K	64K
最高転送速度	Mバイト/秒	4（8MHz版）	1.25	2（2MHz版）
対応デバイス	ビット	8/16	8/16	8
複数ブロック転送機能		アレイ・チェーン，リンク・アレイ・チェーン，コンティニュー	ソフトウェア・プログラマブル	コンティニュー
その他		▷リトライ機能，ホルト機能，バス・エラー機能などの例外処理機能 ▷動作タイミング・エラー，カウント・エラー，コンフィギュレーション・エラーなどのエラー検出機能 ▷アドレス/データ多重	▷I/Oプロセッサとして動作する ▷トランスレート機能 ▷サーチ機能 ▷I/Oアドレス空間とメモリ・アドレス空間を分離 ▷アドレス/データ多重	▷メモリ-メモリ間転送機能なし ▷HMCS6800周辺LSI，ただしHD68000と直接インタフェース可能

命令などの転送命令により，データ転送を実行することができますが，アドレス情報，転送語長および転送後の語長計算をして転送終了をソフトウェアですべて考慮しなければならないため，大量の高速データ転送は見込めません．

そこで，マイクロプロセッサに代わってバス占有権を奪い，この高速データ転送を制御するMC68000周辺LSIであるDMAC（DIRECT MEMORY ACCESS CONTROLLER）が登場します．いままでにも，8ビット・マイクロプロセッサ用DMAコントローラやインテルi8089 I/Oコントローラなどが製品化されていますが，MC68450は，16ビットのDMA転送専用コントローラとしては初めてのものです．入出力機器制御用のプロセッサとして働き，その手段としてDMA転送機能を備えているi8089とは，基本的に異なったLSIです．

各種DMAコントローラの機能比較を表9·1に示します．

MC68450は，MC68000の持つ高信頼性システム設計を反映しており，他の

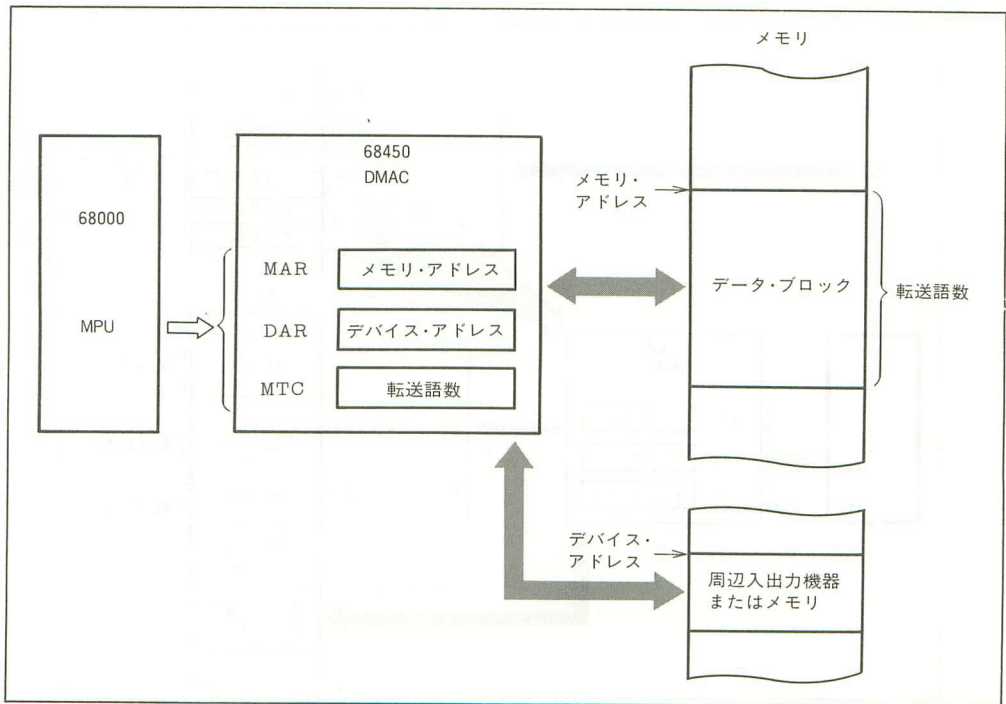



図 9·7 単一ブロック転送

DMAコントローラにはない例外処理，エラー検出機能や，外部アボート入力やソフトウェアによるホルト，アボートなどの機能を備え，しかも最高転送速度4Mバイト/秒を実現し，従来のDMAコントローラの性能を大幅に超えたものといえます．MC68450は，1チップに独立したチャネルを四つ内蔵し，各チャネルの優先順位もソフトで設定できます．

転送データの単位は8，16，32ビットで，32ビットのデータ転送では，二重アドレッシング・モードを用いて実行します．

MC68450は，四つの完全に独立したチャネルをそれぞれ異なる入出力機器を接続し，連続したデータ群（データ・ブロック）の転送を，以下に述べる3種類のモードで実行します．

（1） **コンティニュー・モード**（図9·7参照）：単一ブロック転送をMPUの指示により自動的に繰り返す転送方式．単一ブロック転送とは，メモリおよびデ

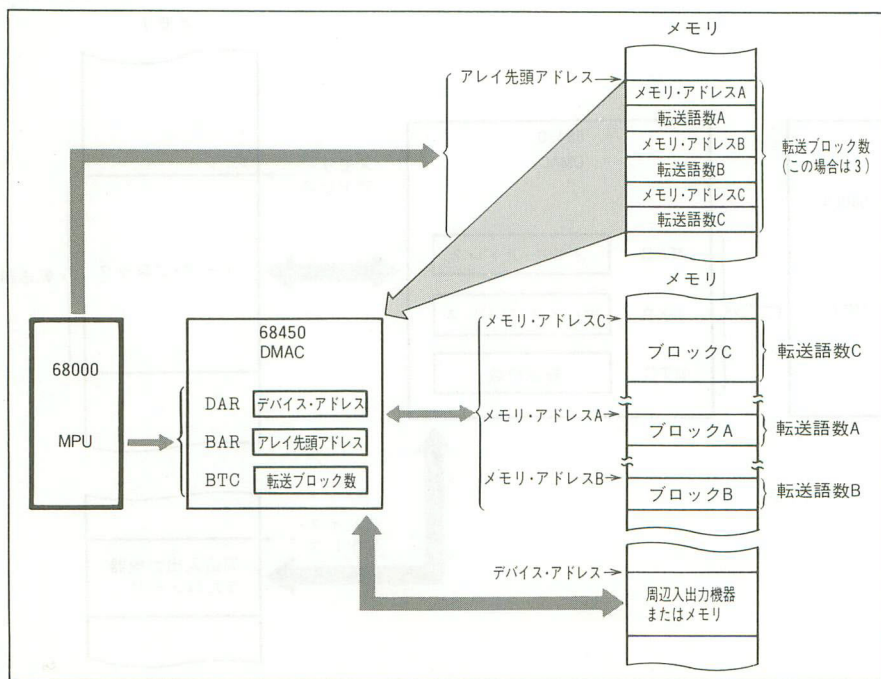



図 9·8 アレイ・チェーン転送

バイスの転送元あるいは転送先の先頭アドレスと転送語数をDMA（内の各レジスタにMPUが設定し，1データ・ブロック転送する方式．

（2）**アレイ・チェーン・モード**（図9・8参照）：MPUがあらかじめメモリ内にブロック情報をアレイ状に並べたテーブルを作成し，DMACの初期化（レジスタ設定）をすることにより，DMACが各ブロック転送ごとにテーブルを参照し，複数のデータ・ブロック転送を自動的に行う方式．

（3）**リンク・アレイ・チェーン・モード**（図9・9参照）：アレイ・チェーンと同様に複数のデータ・ブロック転送に使用されます．両方式ともに各ブロックの先頭アドレスと転送語数をメモリ内に収納していますが，リンク・アレイ・チェーン・モードでは，さらにリンク・アドレス（次のブロックの転送情報の入っているアドレス）も収納しています．したがって，この場合，アレイ・テーブルが連続する必要がなく，ブロック転送の削除や挿入が，リンク・アドレスを書き換え

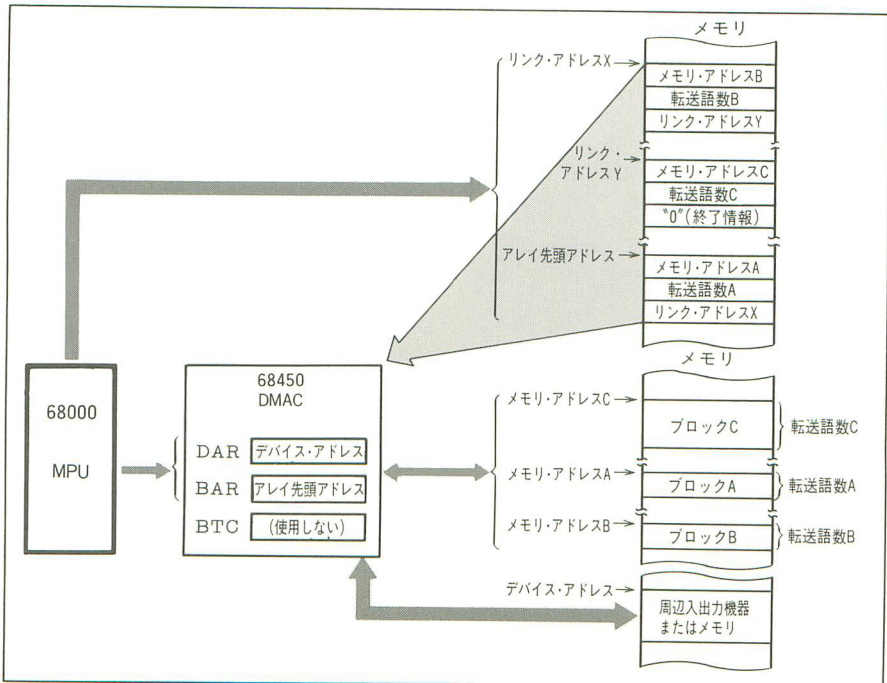



図9・9　リンク・アレイ・チェーン転送

るだけで簡単にできます.

また，DMAC がデータ転送要求を受け付ける方法も，大きく分けて以下の三つがあります.

（1） バースト・モード：連続するバス・サイクルを占有し，連続してデータ転送する方式.

（2） サイクル・スチール・モード：1データ単位ごとにDMAC がMPU からバス占有権を奪いデータ転送をする方式.

（3） オート・リクエスト・モード：DMAC が内部的に転送要求を出す方式.

9・4 MC68000周辺通信用LSI

現在，コンピュータ・システムが扱う情報は，増大/高度化/多様化の方向にあります．そのシステム間の正確な高速データ通信は，情報の停止（腐沈化）を防ぐとともに資源共用化を図ることができ，コンピュータ・ネットワークの信頼性や性能を決める重要な役割を果します．

データ通信方式には，大きく分けて非同期式（調歩同期）通信方式と同期式通信方式の二つがあります．また，同期方式の通信においても，同期方式やデータのフォーマッティングの違いにより，HDLC，SDLC，BISYNC，イーサネットなど各種プロトコルの通信があります．

MC68000ファミリ・データ通信用LSIは，これらの通信方式の一つ以上を満足するように設計されています（図9・10参照）．

また，各データ通信用LSIの機能比較表を表9・2および表9・3に示しました．

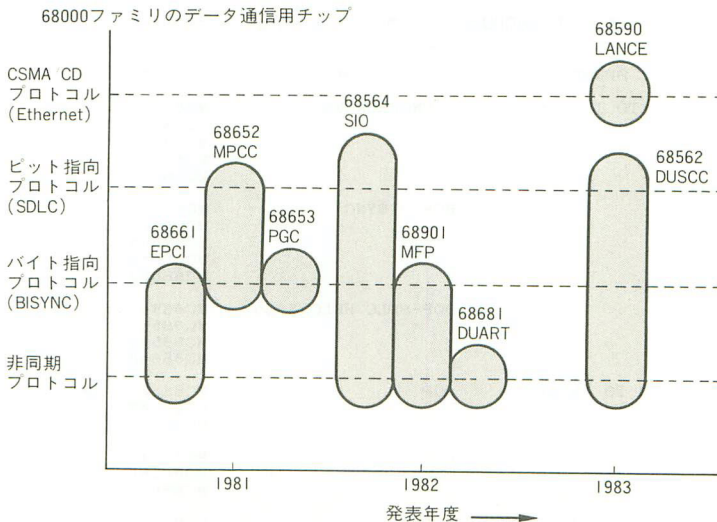



図 9・10　68000ファミリのデータ通信用チップの方向

表 9・2 68000 データ・コミュニケーション・コンパージョン

	68661 EPCI	68681 DUART	68652 MPCC	68564 SIO	68562 DUSCC
68000 INTERFACE	NO	YES	NO	YES	YES
ASYNCHRONOUS	YES	YES	NO	YES	YES
STOP BITS					
1,1.5,2	YES	YES	NO	YES	YES
INCREMENTS	NO	YES	NO	NO	YES
PARITY	YES	YES	NO	YES	YES
SYNCHRONOUS	YES	NO	YES	YES	YES
BOP	NO	NO	YES	YES	YES
BCP	YES	NO	YES	YES	YES
CHANNELS	1	2	1	2	2
MAX RATE(MBITS/S)	1	1	2	1	4
BUFFERING R/T	2/2	4/2	1/1	4/2	4/4
DATA BUS	8	8	8/16	8	8
PROG BAUD RATES	YES	YES	NO	NO	YES
CRC	NO	NO	YES	YES	YES
INTERRUPTS V & AV	NO	V	NO	V & A	V & A
INTERRUPT REGS	0	4	0	2	5
REGISTERS	12	17	4	21	27
PROG TIMER	NO	YES	NO	NO	YES
CHAR LENGTH	5-8	5-8	1-8	5-8	5-8
PROTOCOLS	S=SUPPORTED ON CHIP		C=CAPABLE OF PERFORMING		
BOP					
ACCCP			C	C	C
HDLC			C	S	C
SDLC			C	S	S
X.25			C	C	C
BCP					
BISYNC	C		C	C	S
DDCMP	C		C	C	C
PINS	28	40	40	48	40/48

表 9・3 68000 データ・コミュニケーション・サポート

SPEED	PROTOCOL	DEVICE
0 TO 1M BPS	ASYNCHRONOUS	MC6850 MC68562 MC68564 MC68661 MC68681
	BCP-BISYNC	MC6852 MC68562 MC68564 MC68652 MC68661
	BOP-SDLC/HDLC/X.25	MC6854 MC68562 MC68564 MC68652
1M TO 2M BPS	BCP	MC6852(4.5M) MC68562 MC68652
	BOP	MC6854 MC68562 MC68652
2.0 TO 4.0 BPS	BCP BOP	MC68562 MC68562

9・5 MC68000とMC6846のインタフェース例

MC6846 ROM I/O タイマ（RIOT）と68000のインタフェース例を図9・12に示します．MC6846 RIOTは，2K×8のマスク・プログラムROMと8ビットのI/Oポート，16ビットのプログラマブル・タイマ/カウンタを内蔵した40ピンのパッケージです．

図9・12は，68000とRIOTとのインタフェースを，下位の10本のアドレス・ライン（A1～A10）と16本のデータ・ライン（D0～D15），リード・ライト線（R/$\overline{W}$），リセット（$\overline{RESET}$），イネーブル（E），チップ・セレクト信号を用いたTVBug回路です．

図9・13，図9・14に示すように，RIOTは同期式，非同期式でも68000とインタフェースが行えます．ROMセレクト用とI/Oタイマ用としてCS0上位側とCS1下位側があり，アドレスは，上位側のA6～A10，下位側のA3～A5と，I/Oコントロール・レジスタ用のA0～A3があります．

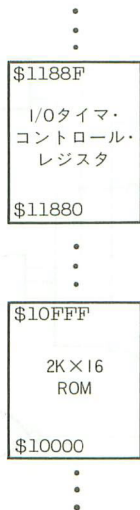



図 9・11 メモリ・マップ

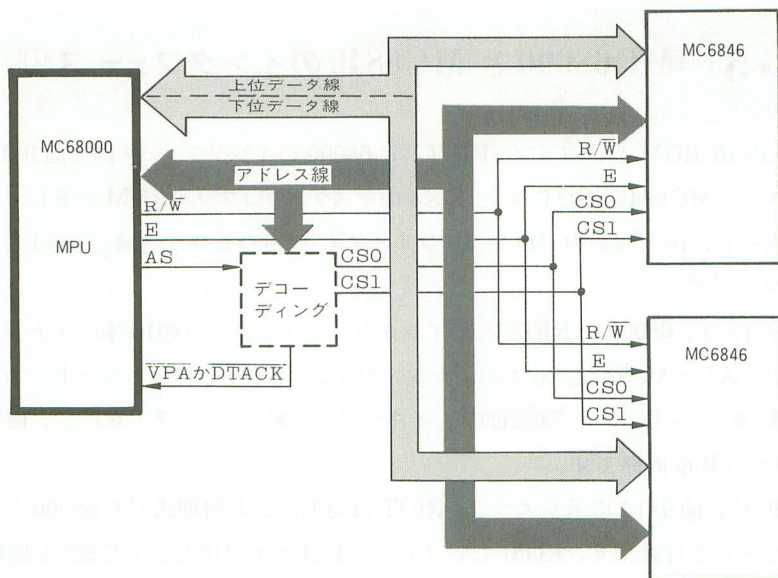



図 9・12　68000とMC6846のインタフェース・ブロック図

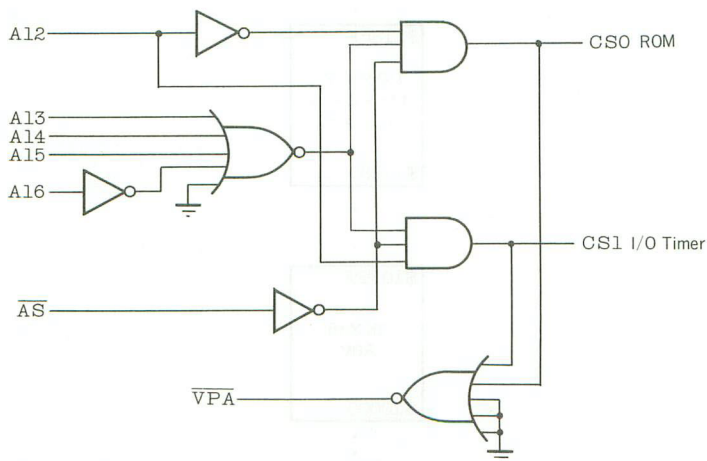



図 9・13　同期式インタフェースのデコーディング回路

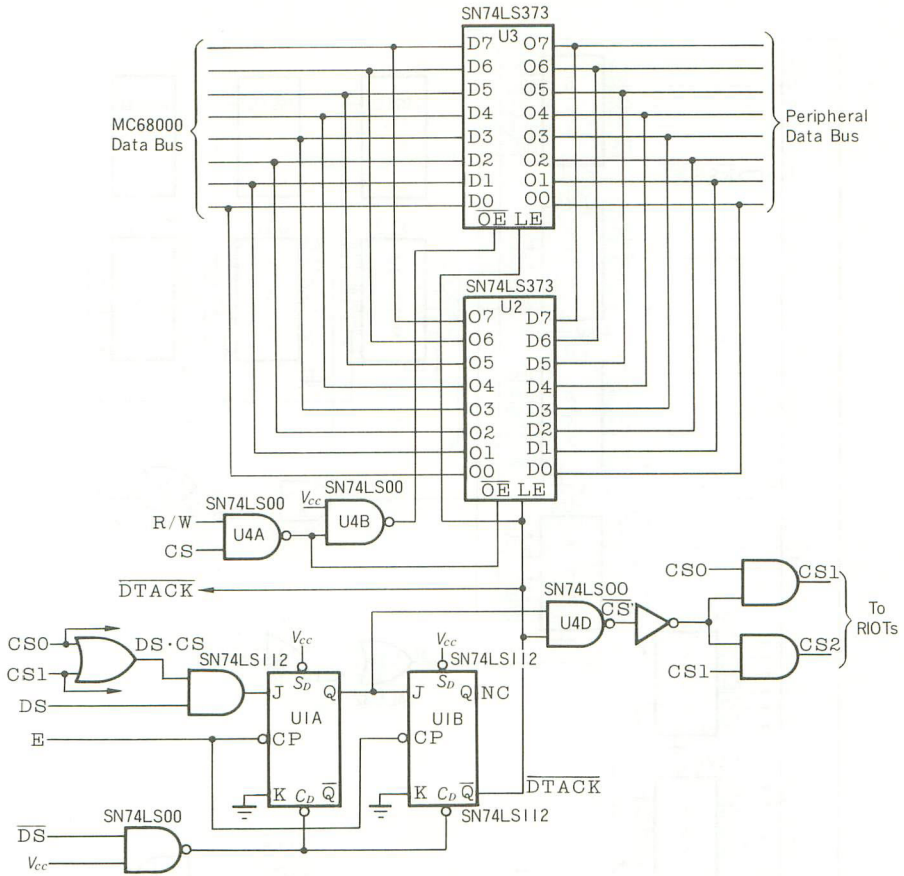



図 9・14 非同期式インタフェース回路

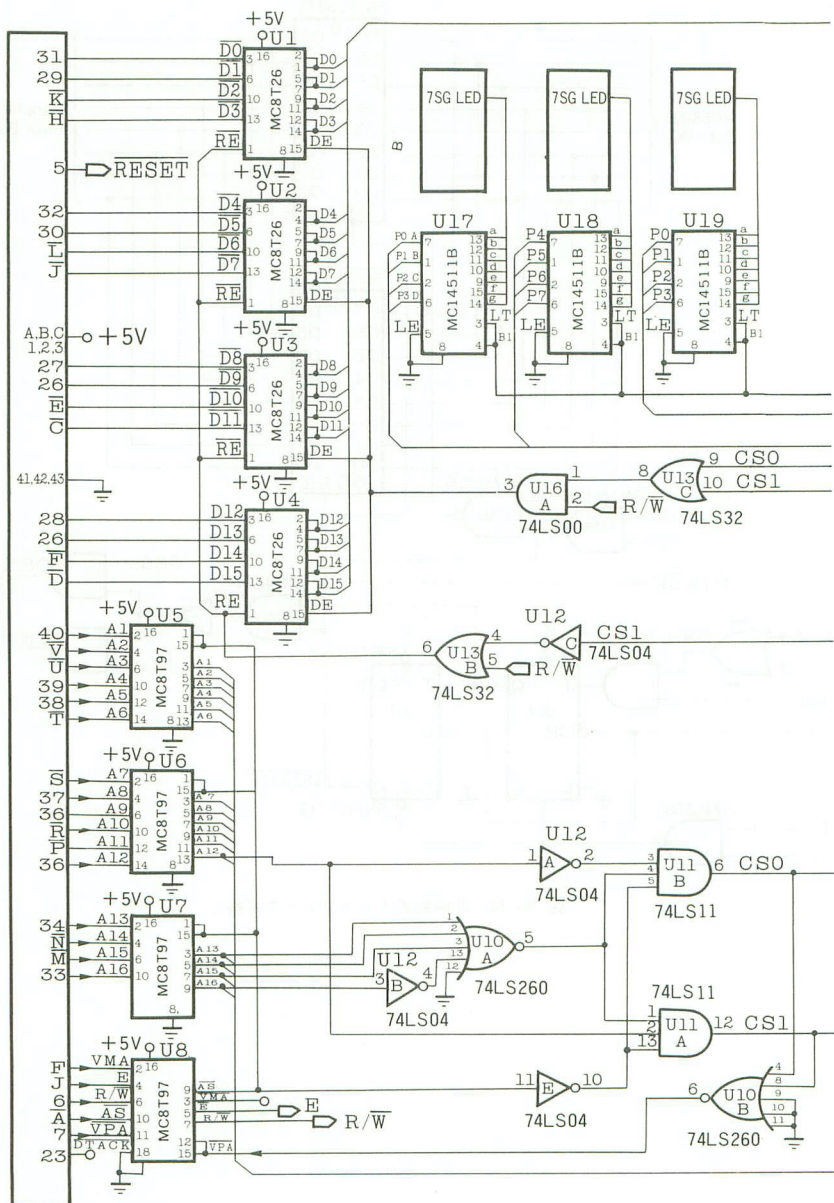
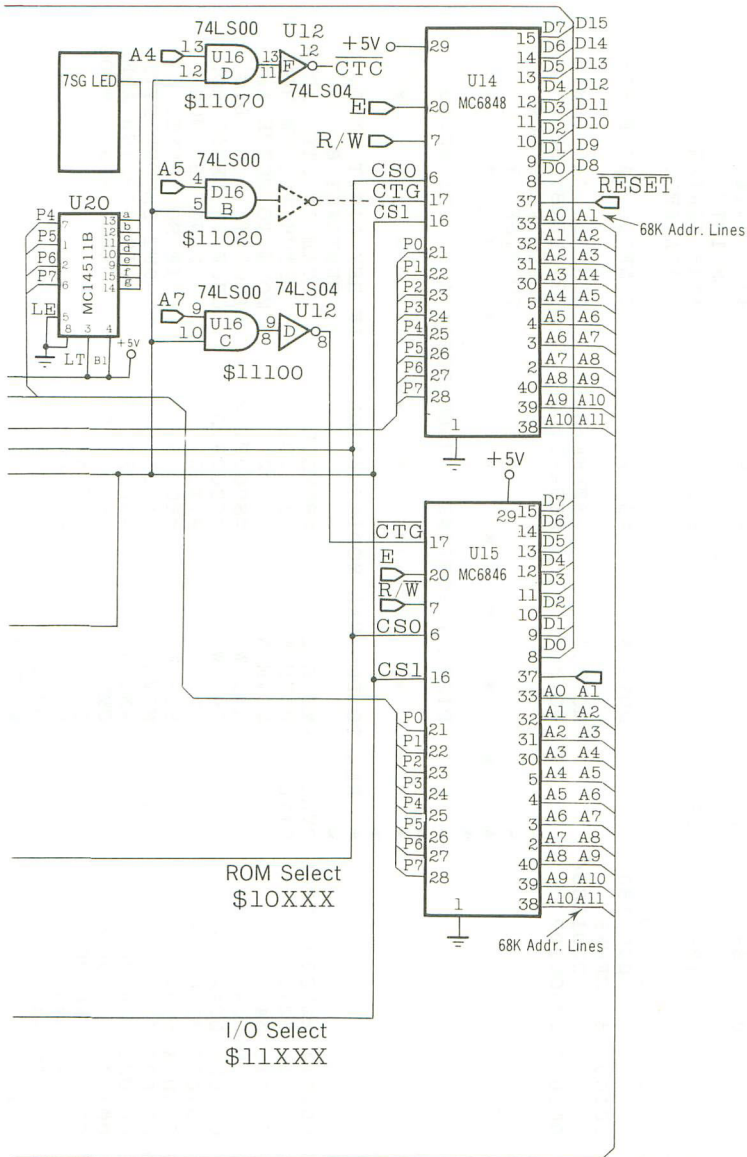



図 9・15　68000 と MC 6846

インタフェースのスケマティック

```
 1          00001000              ORG $1000
 2          00010000       ROMSTR  EQU        $10000       *STARTING ADDR.
 3                         *                                OF '46 ROM
 4          00011882       PCR     EQU        $11882       *'46 PERIPH.
 5                         *                                CNTR. REG.
 6          00011884       DDR     EQU        $11884       *'46 DATA DIR.
 7                         *                                REGISTER
 8          00011886       PDR     EQU        $11886       *'46 PERIPH.
 9                         *                                DATA REG.
10 001000 33FC0000
              00011882             MOVE.W     #$0,PCR      CONFIG. '46 PCR'S
11 001008 33FCFFFF
              00011884             MOVE.W     #$FFFF,DDR   CONFIG. '46 DDR'S
12 001010 303C0009                 MOVE.W     #$0A-1,D0    SET NUMB. OF
13                         *                               WORDS FROM ROM
14                         *                               TO DISPLAY
15                         *    *    *    *    *    *    *    *    *    *    *
16                         *
17                         *       DISPLAY ROUTINE - USES REGS. A1,D0,D1,D2,D3
18                         *
19                         *    *    *    *    *    *   **    *    *    *    *
20                         *
21                         *  HEX TO DECIMAL MODIFICATION--  IF HEX DIGIT GT 9,
22                         *                                SUBTRACT 8
23 001014 227C00010000 CVRCHK  MOVE.L     #ROMSTR,A1        LOAD IN STR. ADDR.
24 00101A 3211         BYTE1   MOVE.W     (A1),D1           FETCH WD. FM ROM
25 00101C 3401                 MOVE.W     D1,D2            MOVE TO SCTCH AREA
26 00101E 0242000F             AND.W      #$000F,D2        ISOLATE L.S. DIG.
27 001022 0C420009             CMP.W      #$0009,D2        CHCK IF NEEDS MOD
28 001026 6F000004             BLE        BYTE2              IF NOT,BRANCH
29 00102A 5142                 SUB.W      #$0008,D2          IF SO, SUBT. 8
30 00102C 3601         BYTE2   MOVE.W     D1,D3             GET SECOND BYTE
31 00102E 024300F0             AND.W      #$00F0,D3          TO SCRATCH AREA
32 001032 0C430090             CMP.W      #$0090,D3           AND REPEAT PROC
33 001036 6F000006             BLE        BYTE3
34 00103A 04430080             SUB.W      #$0080,D3
35 00103E 8443         BYTE3   OR.W       D3,D2
36 001040 3601                 MOVE.W     D1,D3
```

```
37 001042 02430F00             AND.W         #$0F00,D3
38 001046 0C430900             CMP.W         #$0900,D3
39 00104A 6F000006             BLE           BYTE4
40 00104E 04430800             SUB.W         #$0800,D3
41 001052 8443         BYTE4   OR.W          D3,D2
42 001054 2601                  MOVE.L        D1,D3        USE LONG WD. FOR MSB.
43 001056 02830000F000         AND.L         #$F000,D3
44 00105C 0C8300009000         CMP.L         #$9000,D3
45 001062 6F000006             BLE           DONE
46 001066 04438000             SUB.W         #$8000,D3
47 00106A 8443         DONE    OR.W          D3,D2          DISPYD DATA  IN D2
48 00106C 33C200011886         MOVE.W        D2,PDR         WRITE DATA TO PIA
49 001072 2E3C0003D090         MOVE.L        #250000,D7     DLY FOR 5 SEC
50 001078 5387         DLY     SUB.L         #1,D7
51 00107A 6AFC                 BPL           DLY
52 00107C 51C8FF9C             DBRA          D0,BYTE1       GO AGN IF <> 10
53                     *
54                     *         BACK TO MACSBUG
55                     *
56 001080 4E4F                 TRAP          15
57 001082 0000                 DC.W           0
58                     *
59                     * THE PIA ADRESSES ARE   $11882   PCR
60                     *                        $11884   DDR
61                     *                        $11886   TO WRITE TO THE DISPL
62                          END

****** TOTAL ERRORS    0--    0


SYMBOL TABLE

BYTE1       00101A BYTE2      00102C BYTE3      00103E BYTE4      001052
CVRCHK      001014 DDR        011884 DLY        001078 DONE       00106A
PCR         011882 PDR        011886 ROMSTR     010000
```

図 9・16 サンプル回路における 68000プログラム例

MC 68010　仮想マシン

MC 68010 は MC 68000/68008 の機能と特徴を完全にカバーし，Virchual Memory/Machine をサポートします．

68000 と比べての主な追加点と変更点

（1）　二つのレジスタの追加

VBR：ベクタ・ベース・レジスタ

（ベクタ・テーブルをリロケータブルにする）

SFC，DFC：オルタネイト・ファンクション・コードレジスタ

（オルタネイト・アドレス・スペースへの転送）

（2）　新規インストラクションの追加

RTD：Return and Dellocate Stack

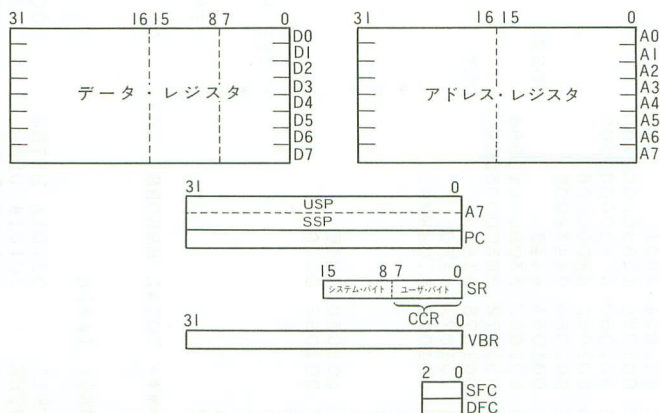
MOVES：Move Alternate Address Space

MOVEC：Move Control Register

MOVE from CCR：Move from Condition Code Register

（3）　1部のインストラクションがスピード・アップされた．

全体的に 20～25％のスピードアップ

（4）　特権命令　　MOVE from SR 命令が特権命令に変更

10. MC 68000開発装置

68000の開発支援装置であるエクサマクス（EXORmacs）とボックス・コンピュータ（VMC 68/2）の構成と機能について，それぞれやさしく図示し説明します．そして，エクサマクスとボックス・コンピュータの特長と用途も説明します．また，用途，方法をより理解していただくために，EXORmacsの実際の使用例，ディスク・オペレーティング・システム（VERSAdos）も載せておきました．

10・1 エクサマクス

エクサマクス（EXORmacs）は，MC 68000（16/32 ビット・プロセッサ）のソフトウェアとハードウェア設計者による開発や設計を容易にするためのサポート・システムです．ハードウェアのアーキテクチャは，今後のマイクロコンピュータにおいて要求されるハイ・パフォーマンスをサポートできます．また，マルチタスクキング・オペレーティング・システムは，フロッピー・ディスク・ベース・システムおよびハード・ディスク・ベース・システムをサポートします．ハード・ディスク・ベースとフロッピー・ディスク・ベースとの相違は，供給されるドライブ・ユニットと **"VERSAdos-E"** オペレーティング・システムによって決定されます．

〔1〕 エクサマクスの特長

（1） MPU に MC 68000 を搭載し，MC 68000 のための最大 8 ステーションのマルチユーザ開発システムです．

（2） マルチタスク・オペレーティング・システムとメモリ・マネージメント機能を持ち，I/O 分散処理方式を備えています．

（3） 構造化文が使えるマクロ・アセンブラと PASCAL 高級言語が使え，シンボリック・デバッギングが可能です．

（4） ハード・ディスク・サブ・システムは，2 台のドライブをサポートし，記憶容量は 32 M バイトから 192 M バイトまで可能です．また，1 M バイトのフロッピー・ディスクをサポートし，2 M バイトまで拡張が可能です．

（5） おのおの 16 M バイトのメモリ空間を持つ一次マップ，二次マップの二重メモリ・マップ機能をサポートし，工業規格のバーサ・バス（VERSAbus）によって，32 ビットのデータ・バスをサポートします．

（6） 自己診断ファームウェアによるセルフ・テストやフロント・パネルで動作状態やエラー箇所が表示されます．

（7） ホスト・コンピュータからオンライン・ダウン・ロードが可能です．

（8） 132 桁のプリンタの駆動ができ，スピードは，180 ch/600 line・s の 2 種

開発製品の仕様書作成
ソフトウェア開発
ハードウェアの製作
ハードウェアの製作
完成
ユーザ製作中の機器
ユーザ製作中の機器
Useモジュール
デバッグ・モジュール
Useモジュール
デバッグ・モジュール
①プログラム作成
②プログラム・デバッグ
③システム・デバッグ
④システム・デバッグ

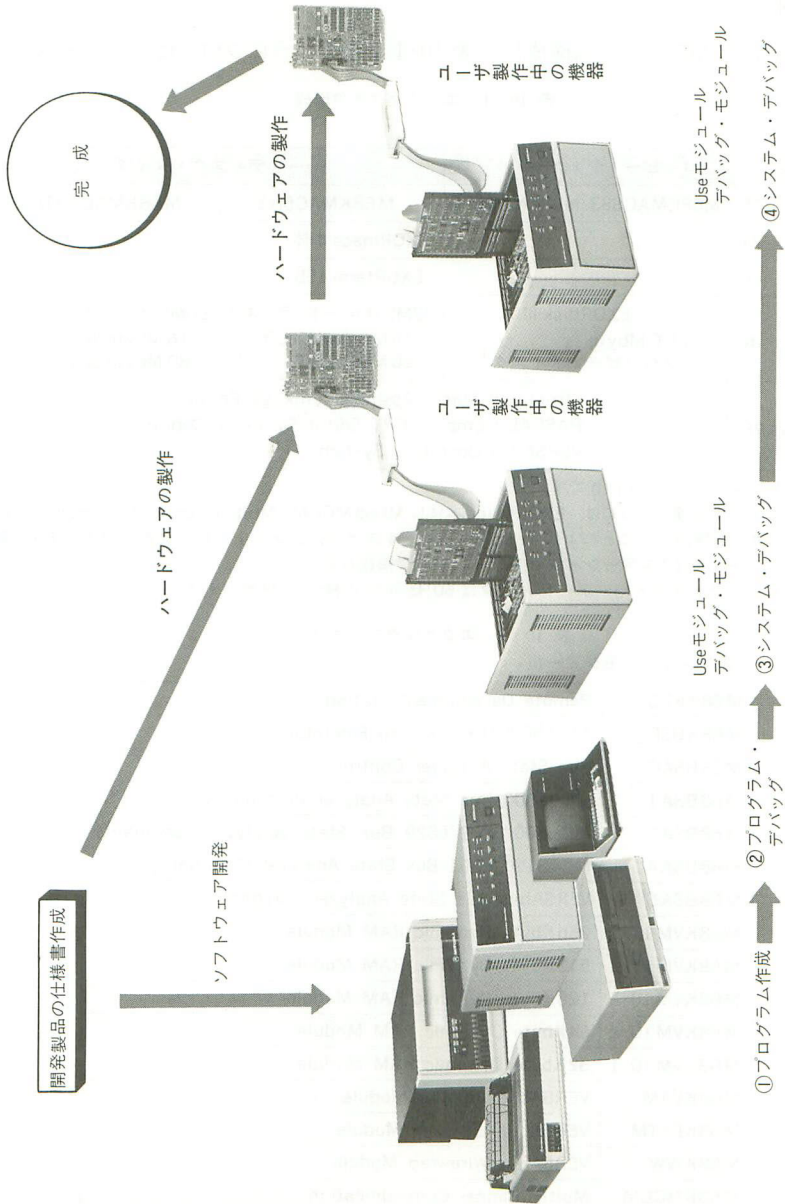



図 10・1 開発のフロー

類のプリンタをサポートし，インテリジェントCRTによってソフト・キー入力ができます．

エクサマクスのシステム構成は，表10·1に示すようにフロッピー・ディスク・

表 10・1 エクサマクスの構成

構 成

	フロッピー・ディスク・タイプ	ハードディスク・タイプ	
タ イ プ	M68KMACSS3/M68KMACSF1	M68KMACSH1	M68KMACSH1A
C P U	EXORmacs本体		
C R T	EXORterm 155		
D I S K	EXORdisk III (1.0 Mbyte フロッピー・ディスク)	32Mbyteハード・ディスク (16 Mbyte取外し可/16 Mbyte固定)	96Mbyteハード・ディスク (16 Mbyte取外し可/80 Mbyte固定)
SOFTWARE	Structured Macro Assmbler/Linkage Editor/ PASCAL Compiler/CRT Editor/Symbolic Debug/ VERSAdos Operating System		

M68KMACSS3はプリンタ付きです．

※マルチ・ユーザ処理を行うには，EXORmacs本体にM68KMCCM（Multi-Channel Communication モジュール）を増設することにより，1モジュールで4ステーションと1プリンタが追加できます（最大3モジュールで12ステーションと3プリンタまで接続可能）．

※フロッピー・ディスクとハード・ディスクは60Hz用と50Hz用に区別されています．

表 10・2 エクサマクスのオプション

オプション・モジュール

M68KRDS	Remote Development Station
M68KUSE	MC68000 User System Emulator
M68BSAC	Bus State Analyzer Control
M68BSA1	MC68000 Bus State Analyzer Personality
M68BSA2	MC6800/6809/6829 Bus State Analyzer Personality
M68BSA4	MC68120/6801 Bus State Analyzer Personality
M68BSA5	VERSAbus Bus State Analyzer Personality
M68KVM11-1	256Kbyte Dynamic RAM Module
M68KVM11-2	512Kbyte Dynamic RAM Module
M68KVM10-3	128Kbyte Dynamic RAM Module
M68KVM10-2	64Kbyte Dynamic RAM Module
M68KVM10-1	32Kbyte Dynamic RAM Module
M68KVAM	VERSAbus Adapter Module
M68KEXTM	VERSAbus Extender Module
M68KWW	VERSAbus Wirewrap Module
M68KMCCM	Multi-Channel Communication

タイプとハード・ディスク・タイプの**2**タイプがあります．また，オプション・モジュールを表10·2に示します．

〔2〕 **EXORmacsの装置**

（1） **シャーシ**は，モジュール・ハウジング，電源，空冷ファン，フロント・パネルなど．

（2） **デバッグ・モジュール**は，EXORterm 155ディスプレイ・コンソールと，ホスト・コンピュータ用の二つのRS-232シリアル・ポートと，703，B600プリンタ用のポートと，合計三つのポートを備えています．

（3） **マイクロプロセッサ**・モジュールは，4セグメントのメモリ管理ユニットと診断用ファームウェアを備え，マルチ・タスク・オペレーティングの保護をします．

（4） **フロッピー・ディスク・コントローラ**・モジュールは，4台までのフロッピー・ディスク・ドライブのサポートをします．

（5） **128Kバイト・ダイナミック・メモリ**・モジュールは，オペレーティング・システムとサポート・ソフトウェアを格納します．

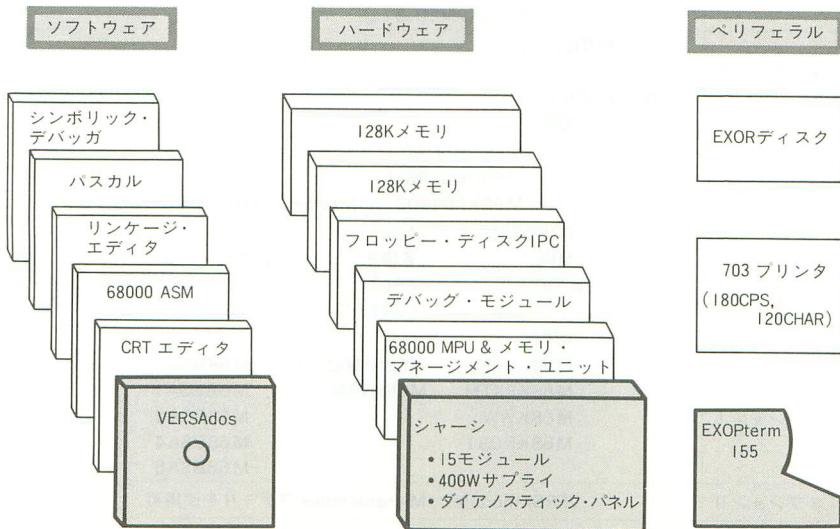



図 10・2 EXORmacsシステムによるハードウェア，ソフトウェア開発装置

（6） **汎用 IPC** モジュールは，ユニバーサル・ディスク・コントローラ（UDC）の一つであり，ハード・ディスクとフロッピー・ディスクを制御するソフトウェアも含んでいます．

（7） **ディスク・インタフェース**・モジュールは，標準 DMA を介して汎用 IPC へのインタフェースを行います．

［3］ **EXORmacs のシステム**

（1） **オペレーション・モード**は，二つのモードを持ち，オペレーティング・

表 10・3　16/32 bit 開発支援セレクション・ガイド

	シングル・ユーザ・システム（2ターミナルで使用可）	4ユーザ・システム	8ユーザ・システム
基本構成	M68KMACSS3 または M68KMACSF1	M68KMACSH1 M68KMCCM×1 M68SXD10155×4 M68KVM11-1×1 M68KFD1102	M68KMACSH1A M68KMCCM×2 M68SXD10155×8 M68KVM11-2×1 M68KFD1102
周　辺	M68K703LP1 M68SXD10155 M68KFD1102		
メモリ	M68KVM10-1　M68KVM11-1 M68KVM10-2　M68KVM11-2 M68KVM10-3		
マルチユーザへの拡張	M68KHDS32-1 または M68KHDS96-1 M68KMCCM		
マス・ストレージの拡張	M68KHDE32-1 M68KHDE96-1 M68KFD1102　M68SFDU1102E		
オプション・ソフトウェア	FORTRAN　ADA MPL*　各種8ビットMPUアセンブラ COBOL*　APL* BASIC-M　FORTH* UNIX*		
オプションI	M68KUSE　M68KMACSRK　M68BSAC M68KEXTM　M68KVAM　M68BSA1 M68KWW　M68BSA2 M68KRDS1　M68BSA4 M68BSA5		
オプションII	VERSAmodules, Micromodules ファミリを使用可		

＊　他社から供給の予定

（注）　フロッピー・ディスクとハード・ディスクは，50 Hz と 60 Hz 用に区別されております．

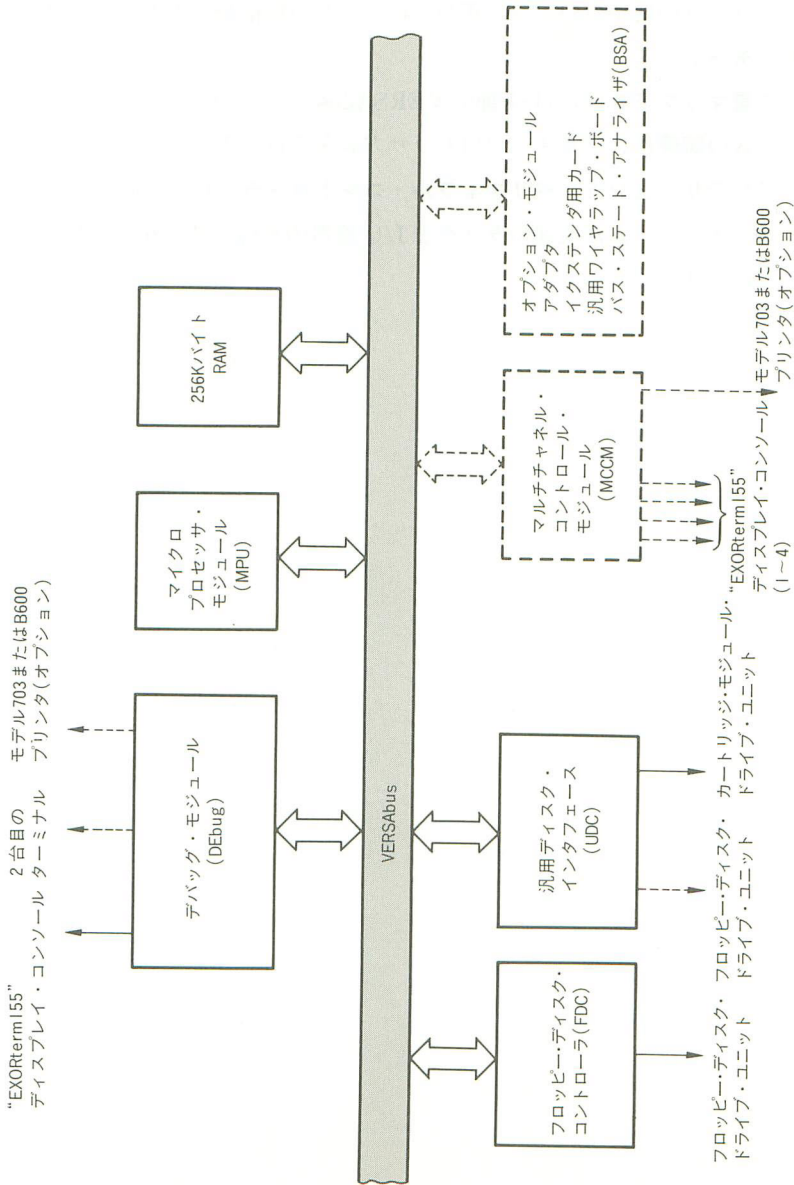



図 10・3 機能別ブロック・ダイアグラム

システム用や，すべてのオペレーションが制限なしに実行できる**スーパバイザ・モード**と，ユーザの応用プログラム用や，いくつかの特権命令の実行に制限のある**ユーザ・モード**です．

（2）　**二重マップ方式**は，11·1 節の VERSAdos ソフトウェア参照．

（3）　**バスの調停**は，マルチ・プロセッサ方式を実行できます．

（4）　**インテリジェント・ペリフェラル・コントローラ**（IPC）により，オペレーティング・システムが現在のタスクを I/O 機器の制御に時間をとられることなく実行できます．

10・2 エクサマクスの使用例

```
例   COMPIL.CF
          =ARG
          =PASCAL \1/\2,,\3          Argl,2で示されるPASCAL・ソース・
          =PASCAL2 \1,,#NULL         プログラムをコンパイルしてリンクの後, ロー
          =LINK \1,,\1;IXM           ド・モジュールを作り出すもの
          =END
(実行)

          =BATC COMPIL QUEENS,NULL;#PR ←── 一括モード処理の開始を指定する
          000B: QUEUED
          =BATC COMPIL QUEENS,NULL,#PR ←── さらに同じ内容の処理を開始させる
          000C: QUEUED '
          =QUER ←──────────────────── "QUER"コマンドによりシステム内のジョ
          000B: RUNNING                     ブの状態をチェックする
          000C: QUEUED
          =CANC ←──────────────────── 最後に登録したジョブを取り消す
          000C: DONE=A006
          =QUER ←──────────────────── "QUER"コマンドにより最後に登録された
          000C: DONE=A006                   ジョブの状態をチェックする
          =

 この一括処理を行った結果として次のファイルが作成される
          SYS:0000..QUEENS.CF
          SYS:0000..QUEENS.SA
          SYS:0000..QUEENS.PC
          SYS:0000..QUEENS.RO
          SYS:0000..QUEENS.LO         作成されたファイル
          SYS:0000..QUEENS.LL
```

図 10・4 コマンド・ファイル(CDF)の内容

```
          =ARG
          =PASCAL \1/\2,,\3
          =PASCAL2 \1,,#NULL
          =/ABT                        ここで使用されるチェーン・ファイル
          =LINK \1,,\1;IXM             COMPIL1.CFは左の通り
          =END
          END CHAIN                           COMPIL1をチェーン処理で実行する
                                              このとき, 引数1,2,3をQUEENS,
          =CHAI COMPIL1 QUEENS,NULL,#CN00     NULL, #CN00と指定する
          =ARG
          \1:QUEENS
          \2:NULL
          \3:#CN00
          =PASCAL QUEENS/NULL,,#CN00
           Motorola Pascal Compiler Phase 1  Version 1.20
           Copyrighted 1981 by Motorola, Inc.

           **** No Error(s) detected in compilation ****
          =PASCAL2 QUEENS,,#NULL
           M68000 Pascal Compiler Phase 2 Version  1.20
           Copyrighted 1981 by Motorola, Inc.

           CODE GENERATOR PRODUCED 000003C2 BYTES OF CODE.
           NO ERRORS DETECTED.
          =/ABT
          RX=$0000 RA=$666C RD=$0000
          CHAIN ABORTED
          (CHAIN)=END
```

図 10・5 チェーン・モード例

"EXORmacs" システム生成用ファイル　SYSCMD.SA

(1)

```
MSG SYSGEN COMMAND FILE FOR HARD DISK, MULTI-USER CONFIGURATION
MSG DEFINE ALL DEVICES
TOTTERM=10 TOTAL NO. OF TERMINALS
TOTPRT=3 TOTAL NO. OF PRINTERS
TOTDSK=8 TOTAL NO. OF DISK DRIVES
NOLTERM=2 NO. OF LOCAL TERMINALS
NOLPRT=1 NO. OF LOCAL PRINTERS
HDUDC0=4 NO. OF HARD DRIVES ON 1ST UDC
FDUDC0=2 NO. OF FLOPPY DRIVES ON 1ST UDC
HDUDC1=0 NO. OF HARD DRIVES ON 2ND UDC
FDUDC1=0 NO. OF FLOPPY DRIVES ON 2ND UDC
NOFD0=2 NO. OF FLOPPY DRIVES ON 1ST FDC
NOFD1=0 NO. OF FLOPPY DRIVES ON 2ND FDC
NOIPCS=2 NO. OF DISK FDC'S AND UDC'S
NOTERM0=4 NO. OF TERMINALS ON 1ST MCCM
NOPRT0=1 NO. OF PRINTERS ON 1ST MCCM
NOTERM1=4 NO. OF TERMINALS ON 2ND MCCM
NOPRT1=1 NO. OF PRINTERS ON 2ND MCCM
NOTERM2=0 NO. OF TERMINALS ON 3RD MCCM
NOPRT2=0 NO. OF PRINTERS ON 3RD MCCM
NOTERM3=0 NO. OF TERMINALS ON 4TH MCCM
NOPRT3=0 NO. OF PRINTERS ON 4TH MCCM
NOMCCMS=2 NO. OF MCCM'S
MSG DEFINE OTHER VARIABLE INFO.
NOTASKS=30 MAX. NO. OF TASKS IN SYSTEM AT ONE TIME
MAXLU=8 MAXIMUM LU FOR EACH TASK
DCQPGE=2 NO. OF PAGES IN DEVICE CONNECTION QUEUE
GST=4 NO. OF PAGES IN GLOBAL SEGMENT TABLE
UST=1 NO. OF PAGES IN USER SEMAPHORE TABLE
TRACE=2 NO. OF PAGES IN TRACE TABLE
IOV=1 NO. OF PAGES IN I/O VECTOR TABLE
NOFILES=50 MAX. NO. OF FILES OPEN AT ONCE
NODIFFIL=50 MAX. NO. OF DIFFERENT FILES OPEN AT ONCE
MODEFVOL=30 MAXIMUM NO. OF DEFAULT VOLUMES
DEFFAB=1 DEFAULT FAB LENGTH
DEFDAT=4 DEFAULT DATA BLOCK LENGTH
MMU=$FE2000 MMU ADDRESS
TIMER=$FEE040 TIMER ADDR.
PANEL=$FE0000 FRONT PANEL ADDR.
MEMEND=$180000 MAXIMUM END OF CONTIGUOUS RAM
MSG CHECK PARAMETER VALIDITY
SUBS VALPAR
ASM VALPAR,,#CN00;-C
PAUSE ABORT SYSGEN IF ANY ERRORS IN ASSEMBLY
MSG GENERATE EXEC PROCESS
PROCESS RMS
STACK=$900 EXEC STACK AREA
STARTRMS=$C00 EXEC BEGINNING ADDR.
RMSFATAL=$C0E FATAL SYSTEM ERROR ADDR.
END EXEC
MEMBEG=* START OF AVAILABLE MEMORY
SUBS IXR
ASM IXR,IXR,#CN00
MSG SET UP FHS TASK
TASK FHS, FHS
STATE='DORM'
SESSION=1
```

任意のメッセージを出力することができる

パラメータを設定することができる

アセンブラを実行することができる

タスクとプロセスの処理を指定することができる

Ⓐ キーボードからの入力を待つこともできる

図 10・6 SYSGEN 例

(2)

```
PRIORITY=$D1
FHSSTR=* FHS LOAD ADDR.
FHSASR=*+2 FHS ASR ENTRY POINT
SUBS FHSLNK.CF
LINK FHSLNK ←——————————————— IOSLNK.CF
END FHS
MSG START OF IOS TASK
TASK IOS,.IOS
PRIORITY=$D1
IOSSTR=* IOS LOAD ADDR. ···················⓪
IOSASR=*+2 IOS ASR ENTRY POINT
SUBS IOSLNK.CF          ···················①
LINK IOSLNK             ···················②
END IOS
MSG START OF IOD TASK
TASK IOD,.IOD
PRIORITY=$D1
IODSTR=* IOD LOAD ADDR.
IODASR=*+2 IOD ASR ENTRY POINT
SUBS IODLNK.CF
LINK IODLNK
END IOD
MSG START OF TTY TASK
TASK TTY,.TTY
PRIORITY=$D4
TTYSTR=* TTY LOAD ADDR.
TTYASR=*+2 TTY ASR ENTRY POINT
SUBS TTYLNK.CF,TTYB
ASM TTYB,TTYB,#CN00
LINK TTYLNK
END TTY
MSG START OF PRT TASK
TASK PRT,.PRT
PRIORITY=$DB
PRTSTR=* PRT LOAD ADDR.
PRTASR=*+2 PRT ASR ENTRY POINT
SUBS PRTLNK.CF,PRTB
ASM PRTB,PRTB,#CN00
LINK PRTLNK
END PRT
MSG START OF IPC TASK
TASK IPC,.IPC
PRIORITY=$D2
IPCSTR=*IPC LOAD ADDR.
IPCASR=*+2 IPC ASR ENTRY POINT
SUBS IPCLNK.CF,IPCB
ASM IPCB,IPCB,#CN00
LINK IPCLNK
END IPC
MSG START OF COM TASK
TASK COM,.COM
PRIORITY=$D4
COMSTR=* COM LOAD ADDR
COMASR=*+2 COM ASR ENTRY POINT
SUBS COMLNK.CF,COMB
ASM COMB,COMB,#CN00
LINK COMLNK ←——————————————— LINKERを実行させることができる
```

①においてIOSLNK.CFの“IOSSTR”にSYSGENユーティリティにより管理されている位置カウンタの値が与えられている
位置カウンタの値は，⓪においてパラメータコマンドにより求められている

図 10・7　IOS TASK 例

(3)

```
END COM
MSG START OF FMS
TASK FMS,.FMS
PRIORITY=$D0
FMSSTR=* FMS LOAD ADDR.
FMSASR=*+2 FMS ASR ENTRY POINT
SUBS FMSLNK.CF
LINK FMSLNK
END FMS
MSG START OF EET
TASK EET,&EET
STATE='READ' ←
SESSION=2 ←――――――― パラメータにはシステム生成ユーティリ
PRIORITY=$C8           ティにより使用されているものもある
EETSTR=* EET LOAD ADDR.
SUBS EETLNK.CF         これらは初期値が設定されている
LINK EETLNK            STATE('READ')
END EET                SESSION(0)
MSG START OF LDR
TASK LDR,&LDR
PRIORITY=$C8 ←
SESSION=4 ←
LDRSTR=* LDR LOAD ADDR
SUBS LDRLNK.CF
LINK LDRLNK
END LDR
MSG START OF IOCOM AND IOI
TASK IOI,.IOI
SESSION=1 ←
PRIORITY=$DA ←
IOCSTR=*
SUBS IOC
ASM DVM/IOC,IOC,#CN00
SUBS ASR
ASM ASR,ASR,#CN00
SUBS IOILNK.CF
LINK IOILNK
END IOI
MSG START OF SYSTEM INITIALIZER
PROCESS INT
INTSTR=*
SUBS IND,INTLNK.CF
ASM IND,IND,#CN00
LINK INTLNK
END INT


=LINK,IOS=#PR,IXHM
SEGMENT .IOS:0 \IOSSTR
INPUT IOS
END
=END
```

図 10・8

AS	ハードウェアを使用したアドレス・ストップ条件を設定します．
BD	ディスク媒体へのメモリ内容のダンプを行います．
BH	ディスク内のデータをメモリにロードし，“MACSbug”にコントロールを移します．
BI	メモリ・ブロックの非破壊的初期化を行います．
BO	セクタ0の情報によりデータのブート・ロードを行います．
BR	ブレーク・ポイント・テーブルへのアドレスの設定を行います．
BT	メモリ・ブロックの破壊的テステを行います（0に初期設定されます）．
CA	ユーザ作成コマンドへのリングを行います．
DC	16進，10進への交換を行います．
DF	トレース表示形式の指定を行います．
DU	メモリの内容をS形式で出力します．
GO	プログラムの実行を指定します．
GT	ブレーク・ポイント（一次的）までのプログラムの実行を指定します．
HE	有効なコマンドの情報を与えます．
LO	S形式のオブジェクト・データを外部デバイスからメモリへロードします．
MD	メモリの内容を表示します．
MM	メモリの内容を変更します．
MS	メモリの内容を設定します．
NOAS	“AS”によりセットされた条件を解除します．
NOBR	“BR”によりセットされた条件を解除します．
NODF	トレースにおける表示形式から一つ以上の項目を削除します．
NOPA	ライン・プリンタ出力を禁止します．
OF	リロケータブル・ファイルのオフセットを表示します．
PA	ライン・プリンタ出力を可能にします．
PF	おのおののポートの特性を設定します．
PR	“DEbug”モジョール・ハードウェアをプライマリ・マップ（一次マップ）に設定します．
RM	各レジスタの内容を変更します．
SE	“DEbug”モジュール・ハードウェアをセコンダリ・マップ（二次マップ）に設定します．
T or TR	プログラムの実行を1インストラクションごとに行います．
TM	HOSTから直接アクセスできるように“DEbug”モジュールのACIAポートを接続します．（PORT1とPORT2は同じボーレイト）
TT	一次的なブレーク・ポイントまでトレースします．
VE	外部デバイスからのS形式のオブジェクト・データと現在のメモリ内容を比較します．
※	ポート2にメッセージを転送します．
·Aφ-·A7	Aφ-A7の内容を表示又設定します．
·Dφ-·D7	Dφ-D7　〃
·Rφ-·R6	Rφ-R6　〃
·PC	PC　〃
·SR	SR　〃
·SS	SS　〃
·US	US　〃

図10・9　MACSbug命令一覧

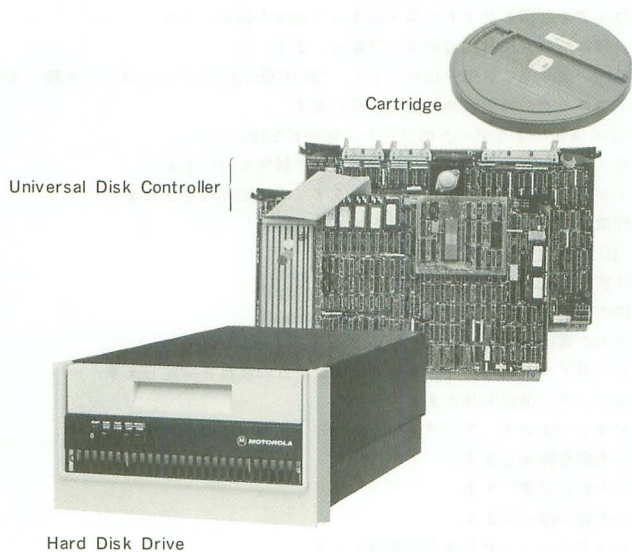



図 10・10　M 68KHDS 32 基本モジュールの構成

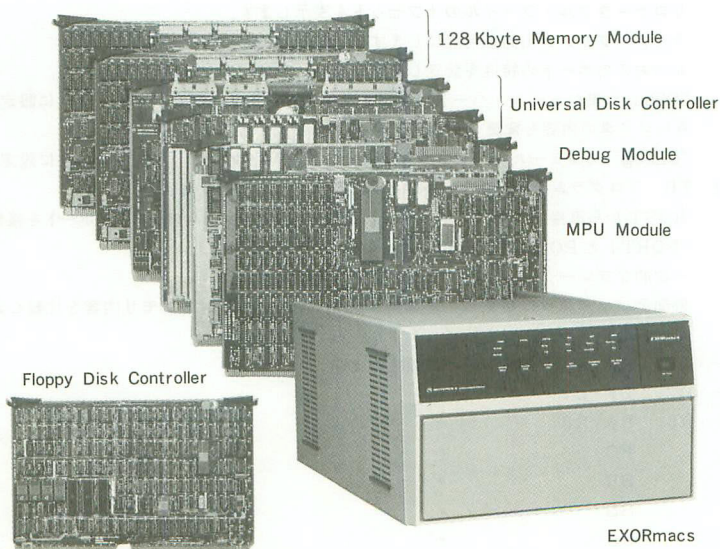



図 10・11　EXORmacs 基本モジュールの構成

10・3 VMC 68/2 マイクロコンピュータ

VMC 68/2は，アプリケーションの基本要素をすでに実装済みで，ユーザが自分の仕様に合わせたボードを選択して追加し，アプリケーション・プログラムをVMC 68/2上で作成し，デバッグ，シミュレート，不要要素を除いてシステム・ソフトウェアに組み入れることで，リアルタイム・マルチタスクのアプリケーションを完成させ，そのまま実際の応用機器に使用できます．したがってVMC 68/2は，多様な仕様に合致させられるような仕様であり，かつ開発期間を短縮することのできる組込み用OEM向けの開発支援装置です．

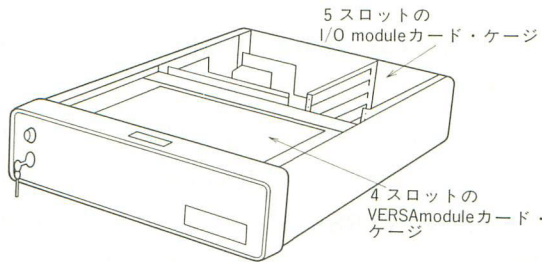



図 10・12 VMC 68/2 MC 68000マイクロコンピュータ・システム

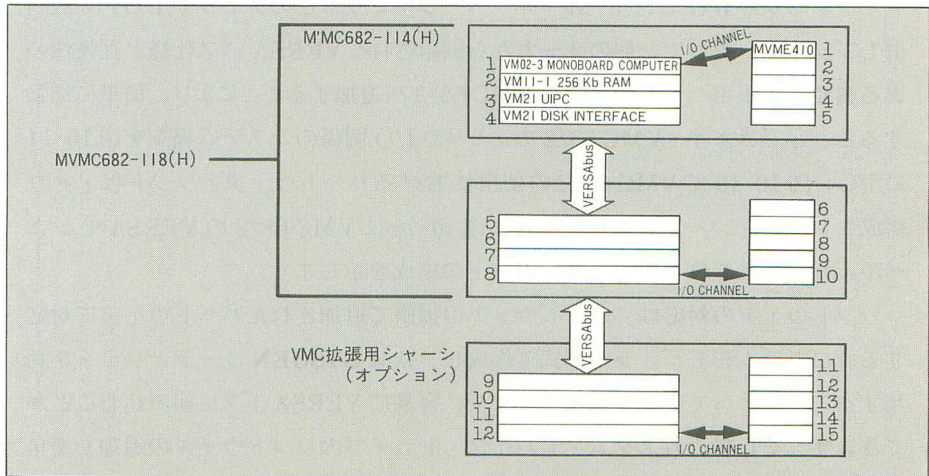



図 10・13 VMC 68/2の構成

VMC 68/2 の構成は，次の三つの開発要素から構成されています．

（1） 処理装置と内蔵基本ボードなどの基本要素

（2） VERSAmodule，I/O Module，パッケージ・ハードウェアなどの拡張要素

（3） エディタ，アセンブラ，リンカ，デバッガ，システム生成ユーティリティとファイル装置などの開発要素

VMC 68/2 の特長は，次の 4 点が代表的なものです．

（1） MC 68000（16/32 ビット MPU）を搭載し，UERSAdas リアルタイム・マルチタスキング・オペレーティング・システムにより高い制御能力を持ちます．

（2） メモリ・マップド I/O 方式により，I/O，メモリ，大容量ファイル装置などの拡張が容易です．

（3） 34 K バイトの内蔵メモリを備え，16 M バイトの直接アドレス空間を持ちます．

（4） デバッガ，単一ライン逆アセンブラ，自己診断，ディスク・ブート・アップ/ダウン・ロードなどのバーサバグ（VERSAbug）のファームウェア．

VMC 68/2 の基本構成は，VERSA モジュール・ファミリから構成されており，VMC 68/2 へ豊富な拡張性を与えています．つまり，VESA モジュール・ボード・ファミリや I/O モジュール・ボード・ファミリなどのメモリや I/O，周辺制御モジュール，そして，他のメーカから供給される VERSA バス仕様と互換性のある製品に拡張用ハードウェアを VMC 68/2 へ追加することにより，簡単に拡張することができます．VMC 68/2 のメモリや I/O 関係のシステム概要を図 10·14 に示し，図 10·15 に VMC 68/2 の使用におけるハードウェアとソフトウェアの構成セレクション・チャートを示し，図 10·16 に VMC 68/2 の VERSA モジュール・カードと I/O モジュール・カードの構成を示します．

ソフトウェアの対応は，ハードウェアの拡張で追加されたハードウェアに対応するプログラムをモジュラー形式で作成し，次に **SYSGEN** ユーティリティを利用することによって実現できます．また，容易に VERSA ドスに組み込むことができます．そして，カスタム・デバイス・ドライブのソフトウェアの追加も簡単に行うことができます．

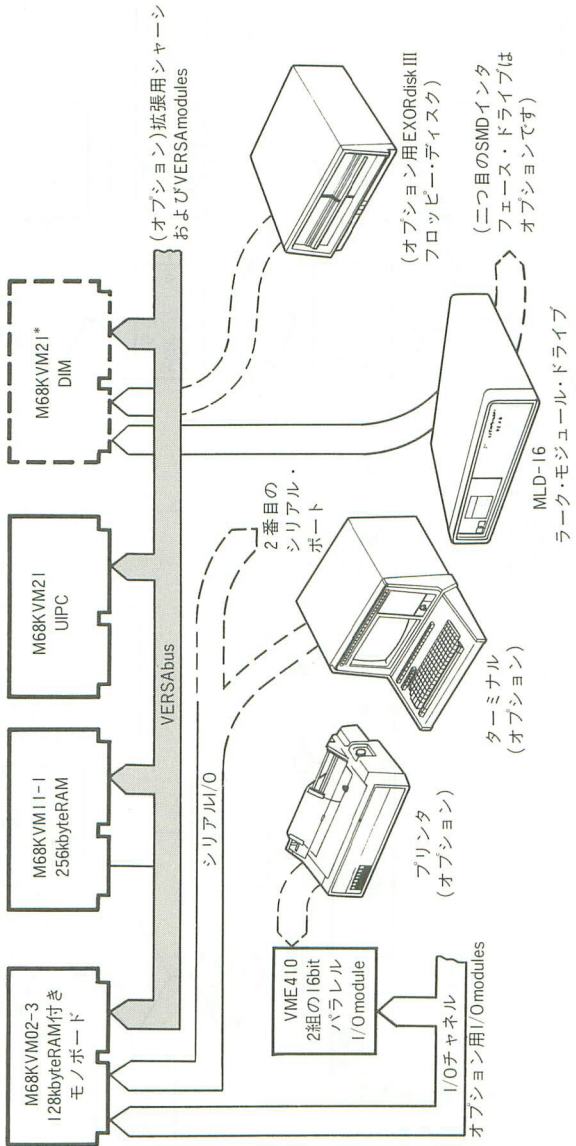
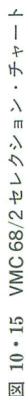



図 10・14 VMC 68/2のシステム概要

図 10・15 VMC 68/2 セレクション・チャート

例として，VMC 68/2 の演算制御用の VERSA モジュールのモノボード・コンピュータ（M 68 KVM 02）がありますが，このマイクロコンピュータ・モジュールは，全般的な機能の作成および周辺機器制御の要求に応ずるために，I/O チャネル・インタフェースを備えています．そして I/O チャネルは，最高 15 枚の I/O モジュールを取り付けることができます．そして，この M 68 KVM 02 は，128 K バイトの双方向 RAM を持ち，マルチプロセッサ構造を備えています．

システム・パッケージは，MLD-16 ディスク・ドライブと VERSA ドス・リアルタイム・マルチタスキング・オペレーティング・システムを含みます．VERSA ドス・オペレーティング・システムは，OEM あるいはシステム作成者が最終的に使用する VERSA ドス構造のアプリケーションを作り上げるために使用するシステム生成ユーティリティを持ちます．この OS は，フォワードとバックグラウンドでバッチ処理ができ，コマンド・プロセッサは，コマンドのチェーンでファイルからのコマンド実行，待ち行列タスクの実行ができます．

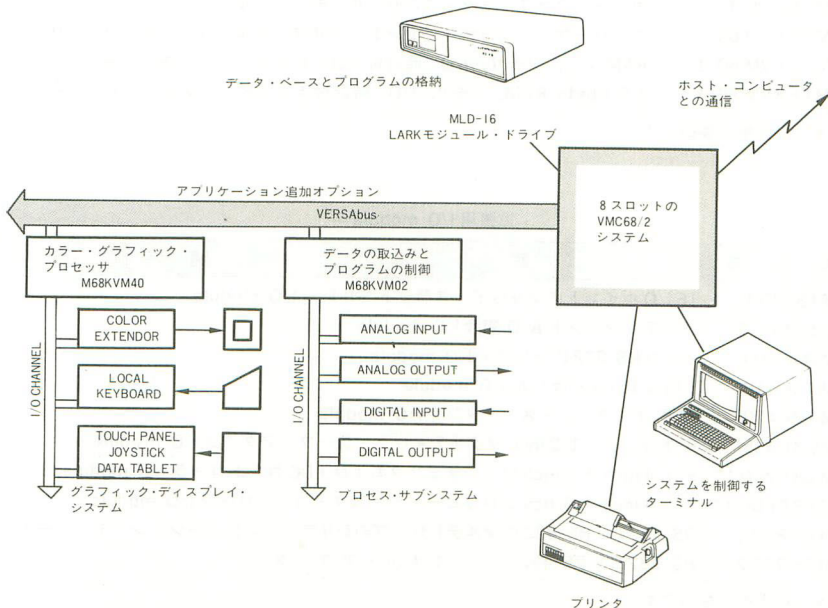



図 10・16　VMC 68/2 を中心として拡張用モジュールを追加したアプリケーション

VMC 68/2 の拡張用 VERSAmodule

製　品　名	説　　明
M 68 KVM 02-3	128 Kbyte の双方向 RAM 付きモノボード・マイクロコンピュータ
M 68 KVM 10-2	64 Kbyte RAM モジュール
M 68 KVM 10-3	128 Kbyte RAM モジュール
M 68 KVM 11-1	ECC 付き 256 Kbyte RAM モジュール
M 68 KVM 11-2	ECC 付き 512 Kbyte RAM モジュール
M 68 KVM 21	ユニバーサル・ディスク・コントローラ（UDC）
M 68 KVM 20	フロッピー・ディスク・コントローラ（FDC）
M 68 KVM 30	4-チャネル・コミュニケーション・モジュール（MCCM）
M 68 KVM 31*	8-チャネル・コミュニケーション・モジュール
M 68 KVM 40	カラー・グラフィック・プロセッサ・モジュール
M 68 KVM 60	ユニバーサル・インテリジェント・ペリフェラル・コントローラ（UIPC）
M 68 KVM 80-1	RAM なし，メモリ，I/O，時計付きコンビネーション・モジュール
M 68 KVM 80-4	128 Kbyte RAM，メモリ，I/O，時計付きコンビネーション・モジュール

＊　計画中の製品です

拡張用 I/O module

製　品　名	説　　明
M 68 RIO 1	16 I/O ポイントのソリッド・ステート・リレー I/O module
M 68 RAD 1	インテリジェント A/D 変換 I/O module
MVME 400	2 組の RS-232 C シリアル I/O module
MVME 410	2 組の 16 bit パラレル I/O module
MVME 420	SASI インタフェース・アダプタ I/O module
MVME 435*	9-トラック，1/2 inch マグネティック・テープ・アダプタ I/O module
M 68 RWIN 1*	5-1/4 inch と 8 inch ウィンチェスタおよび FDC コントローラ I/O module
M 68 RFDC 1*	5-1/4 inch と 8 inch フロッピー・ディスク・コントローラ I/O module
M 68 RSC 1	RS-232 C〜RS-422 のマルチドロップのシリアル・コンバーション・モジュール
M 68 RSC 2	RS-232 C〜RS-422 のターミナル・アダプタ

＊　計画中の製品です

11. 開発ソフトウェア/リアルタイム・モニタ

従来のアセンブラは，生産性や記述性やドキュメント性が低く，柔軟性にも欠けていた．この点を補う方法として，上位言語が用いられるようになってきました．68000 VERSAdos 開発ソフトウェアに含まれる標準ソフトウェアの種類と内容を説明し，開発装置の特長の一つである二重マップ方式を説明します．また，RMS 68 K マルチタスキング・ソフトウェアは，システム支援の中心的機能ともいえるタスク管理の種類と機能について説明します．

11·1　VERSAdos 開発ソフトウェア

VERSAdos は，標準開発ソフトウェアとして用意した設計者の機器開発のツールです．標準装備のソフトウェアとして，CRT エディタ，PASCAL コンパイラ，ストラクチャード・マクロ・アセンブラ，シンボリック・デバッガ・リンケージ・エディタがあります．オプションとしてのプログラムは，FORTRAN，MPL，BASIC-M，COBOL，C/UNIX，ADA，ADLFORTH などの言語をサポートします．

図 11·1 に開発サポートの基本構成を示します．

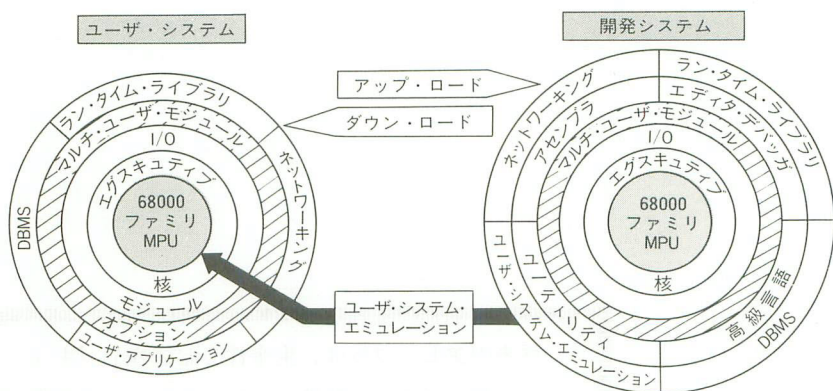



図 11・1　ユーザ・システム・アプリケーション開発装置

CRT エディタは，オペレーティング・システムの監督下で働き，CRT 機能を有効に使用し，CRT 内部でソース・プログラムを効率良く作成や修正ができます．

ストラクチャード・マクロ・アセンブラは，ソース・ステートメントをリロケータブル・マシン・コードに変換し，メモリ・ロケーションを命令およびデータに割り当て，プログラマが指定したアセンブラ作業を行い，随意にクロス・リファレンス・リストを作ります．また，ストラクチャード・プログラミング構成に加えて，マクロおよびコンディショナル・アセンブリの機能も備えています．

```
VERSAdos  VERSION:  n.nn  2/26/80
ENTER DEFAULT SYSTEM VOLUME: USER NO. =SYSO:0<CR>
ENTER DATE (MM/DD/YR) =2/27/80<CR>
ENTER TIME (HR:MIN) =7:46<CR>
7:46:01 2/27/80 START SESSION 0001 USER 0
=DIR *.*<CR>
SYSO:0000..VERSADOS.SY
SYSO:0000..DIR.LO
SYSO:0000..DEBUG.LO
SYSO:0000..DEL.LO
SYSO:0000..COPY.LO
SYSO:0000..FREE.LO
SYSO:0000..MIGR.LO
SYSO:0000..MBLM.LO
SYSO:0000..DUMP.LO
SYSO:0000..BACKUP.LO
SYSO:0000..FMSTST.LO
SYSO:0000..E.LO
SYSO:0000..INIT.LO
SYSO:0000..RENAME.LO
SYSO:0000..SCRATCH.LO
SYSO:0000..SMLOAD.LO
SYSO:0000..PATCH.LO
SYSO:0000..LIST.LO
```


図 11・2 VERSAdos の例

```
1.              LLNT 80
2.              TTL  SORT
3.        *     DEMO PROGRAM  SORT
4.        *
5.              ORG          $1000
6.        START      BRA      PROC
7.        PROC       MOVE.L     #$1000;A7
8.                   MOVE       #NOENT,DO
9.                   PEA        TABLE
10.                  CLR.L      D2
11.                  CLR.L      D3
12.                  JSR        SORT
13.                  STOP
14.       *
15.       *          SORT PROCEDURE
16.       *
17.       SORT       MOVE.L     4(A7),A6
18.                  MOVE.L     DO,D1
19.       SLOOP      DIVS       #2.D1
20.                  AND.L      #$FFFF,D1


F1        F2         F3        F4       F5       F6         F7
CRT       SCROLL     PAGE      PAGEV    LINE^    LINEV      DUP          FILE.SA
```

図 11・3 CRT エディタ

PASCALコンパイラは，プログラミング技術や手法を促進する高度に構造化された言語であり，プログラム記述の容易さやドキュメント性に優れており，プログラムの製品性を高めます．

FORTRANコンパイラは，ANSI，FORTRANワクランゲージの仕様をしのぐものであり，ISAイクステンションと互換性のあるリアルタイム・プロセッシング機能を備えています．そしてROMableで，リロケータブルなオブジェクト・コードを作成します．

リンケージ・エディタは，コンパイラやアセンブラなどの異なった言語を使用して出力した個々のリロケータブル・オブジェクトを結合し，一つのロード・モジュールを作ります．この作業において，セグメント・アトリビュートの指定，アドレス空間の計算，ライブラリの検索，外部参照ラベルの定義，オブジェクト・

```
* TRACE 3 ASSEMBLY INSTRUCTIONS
  $001000 43F81026        LEA      DATA,A1
  $001004 2049            MOVE.L   A1,A0
  $001006 2610            MOVE.L   (A0),D0

* DISPLAY MEMORY   ?A1   ?A0;2   (A0);4   ?D0;3
    ?A1=$00001026  ?A0=$04134   (A0)=$000076A9  ?D0=$0076A9

* DISPLAY SYMBOL   DATA
    DATA=$1026
```

図 11・4 シンボリック・デバッグ（ASSEMBLER）

```
*  TRACE 1 PASCAL INSTRUCTION
   IF long[1] > 255 THEN

* SET MEMORY long[1] 260

* TRACE 1 PASCAL INSTRUCTION
  FOR i :=1 TO 4 DO

* TRACE DISPLAY i long[i] carry temp

* TRACE 3 PASCAL INSTRUCTIONS
  temp := long[i] + carry;
                               i=1  long[i]=260  carry=0  temp=260
  carry :=temp DIV 256;
                               i=1  long[i]=260  carry=1  temp=260
  long[i] := temp MOD 256
                               i=1  long[i]=4    carry=1  temp=260
```

図 11・5 シンボリック・デバッグ（PASCAL）

コードのリロケート，エラー・メッセージの表示を行います．さらに，モジュール・マップや外部定義シンボル・テーブル，未定義や二重定義シンボルをレポートします．

シンボリック・デバッガは，ランゲージ・プロセッサやリンケージ・エディタで作成されたプログラムを，SYMbug によってステートメント・レベルでデバッグします．ステートメント，数値，ブロックのようなプログラム・エレメントをシンボリックに調べて修正します．ブレーク・ポイントのプログラムへの挿入やブレーク要求，あるいはデータの変更を含めて，シンボリックに実行をコントロールします．特殊なエレメントとサブ・エレメントをシンボリックに検索します．ユーザ定義のマクロ・コマンドの作成ができ，指示されたプログラム位置からトレースが実行できます．

```
        M68000 ASSEMBLY LANGUAGE PROGRAM WHICH FINDS
        THE GREATEST VALUE IN ARRAY,DATA , LEAVING
        THAT VALUE IN REGISTER D3 AND THE ADDRESS OF
        ITS FIRST OCCURRENCE IN REGISTER A0

        ORG       $1000
        OPT       BRS,FRS                       SHORT FORWARD REFERENCE OPTIONS
        LEA       DATA,AI                       LOAD BASE ADDRESS INTO AI
        MOVE.L    AI,A0                         A0 STARTS WITH FIRST ELEMENT
        MOVE.L    (A0),D3                       D3 STARTS WITH FIRST ELEMENT

        FOR.L DI=#4 TO #4*(SIZE-I) BY #4 D0     : DI IS USED AS AN INDEX REG
           IF.1 (AI,DI)   GT  D3
             THEN BEGIN                         : SEARCH IS SUCCESSFUL
                    LEA      (AI,DI),A0         : LOAD NEW ADDRESS
                    MOVE.L   (A0),D3            :LOAD NEW VALUE
                 END

*
SIZE    DC        $20
DATA    DS.L      SIZE
        END
```

図 11・6 ストラクチャード・マクロアセンブラ

ソフトウェア開発における全体の流れ図を，図 11·7 に示します.

開発用アプリケーション・プログラムの使用方法として，M 68000 EXORmacs 開発システムを用いて，M 68000 CRT エディタにてアセンブリやデバッギングを行ないます.

文字の入力は，EXORterm 155 キイボードを用います．また，プログラム・ファイルの入力には，ディスク上の EXORmacs オペレーティング・システムである VERSAdos より行ないます.

EXORmacs 開発システムは，**SYMbug**，**DEbug**，**MACSbug** の 3 種類のモニタ/デバッグを備えており，それぞれプログラムのローデングやメモリのテストおよびプログラムの実行ができます．また，MACSbug は，シングル・ユーザのみに用いられますが，SYMbug と DEbug は，VERSAdos のもとにマルチ・ユーザにて用いることができます.

デバッキングのフォーマットは，次のようになっています.

MACSbug コマンド・フォーマット
　P*[N o]<コマンド>[<パラメータ>][; <オプション>]

SYMbug コマンド・フォーマット
　SYMbug [<タスク>] ? [N o] <コマンド> [<パラメータ>] [; <オプション>]

DEbug コマンド・フォーマット
　DEbug [<タスク>] ? [N o] <コマンド> [<パラメータ>]

VERSAdos コマンド・フォーマット
=<コマンド> [<入力フィールド>] [, <出力フィールド>] [; <オプション>]

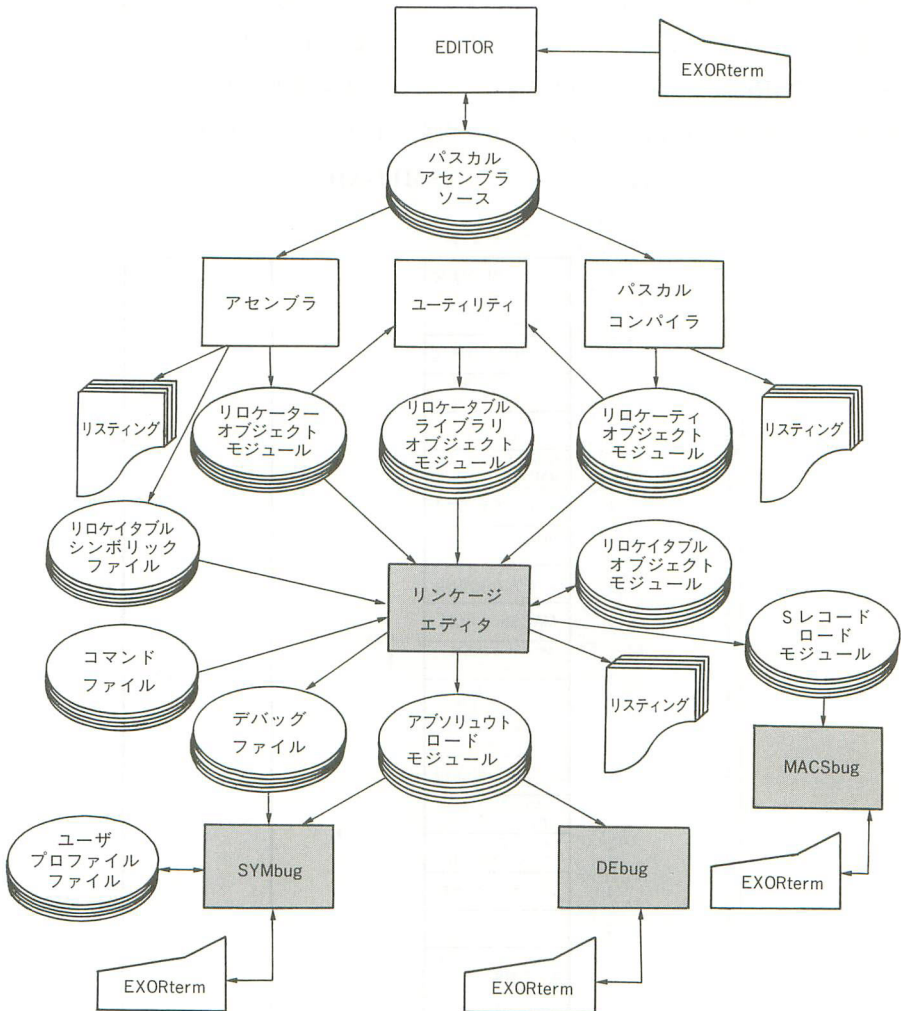



図 11・7 開発プロセスにおけるソフトウェアの流れ

EXORmacsの二重マップ方式

EXORmacsのメモリ・マップは，図11·8に示すように，メモリ・マップは一次と二次の二重マップ方式をとっています．一次マップは，プログラムはプログラム開発用エリヤとし，二次マップは応用プログラムのエミュレーションのエリヤとして，EXORmacsシステムのエミュレーションや開発能力へのフレキシビリティを持たせ，最終的な設計や開発において拘束されることなく実行できることです．

デバッグ方法は，図11·8に示すように**SMLOAD**プログラムをロード・モジュ

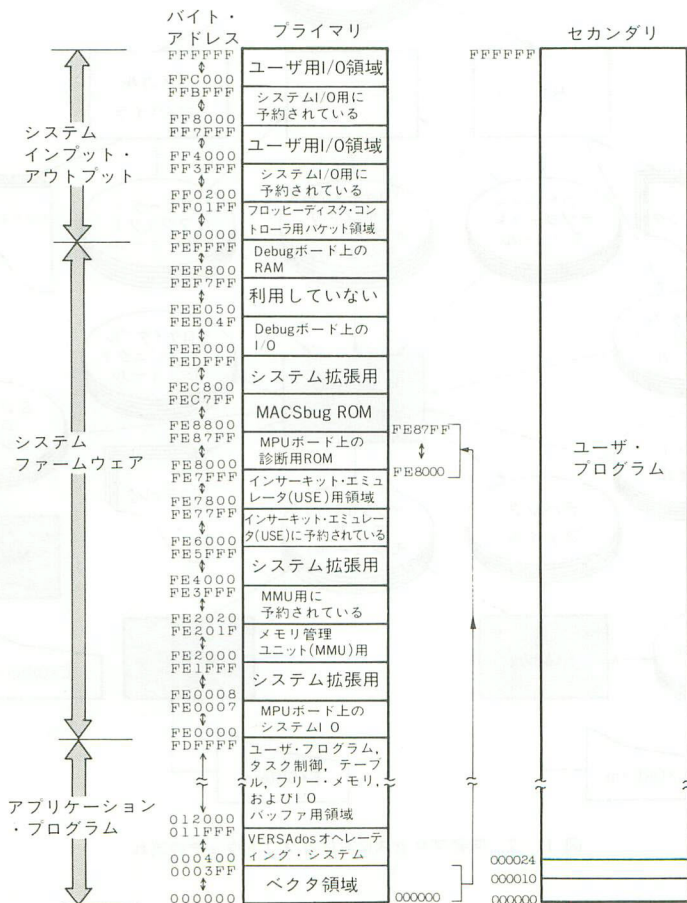



図 11・8　EXORmacsのメモリ・マップ

ールの二次マップ・メモリにロードし，**MACSbug** によってデバッグが行われます．

MACSbug は，一次マップでも二次マップでも実行が可能です．図 11·9 に，一次マップから二次マップへの切換えフロー図を示します．また，図 11·10 にメモリ・**二重マップ**を示します．

二重マップ方式は，ユーザに対して“EXORmacs”のサポートおよび MC 68000 マイクロプロセッサの制限のない使用手段を供給するためのものであり，一次マップにおけるすべての領域では，すべてのインストラクションが実行でき，どのメモリにもアクセスできます（$10 と $24 は除く）．また，二次マップ使用中は，一次マップに格納されているシステム・ソフトウェアおよびシステム I/O は，ユーザのプログラムと完全に分離されます．

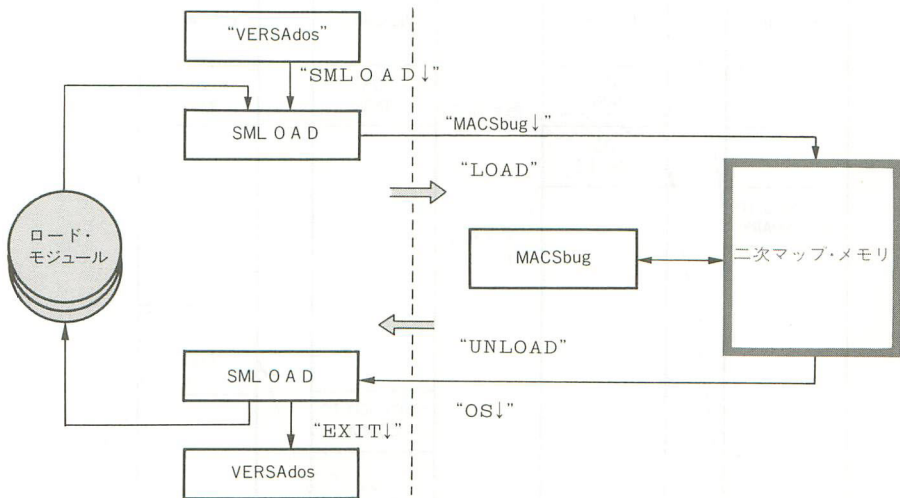



図 11・9　二次マップの実行フロー図

図 11・9 のフロー図補足

"SMLOAD" は，ロード・モジュールの LIB（ローダ・インフォメーション・ブロック）に従い，**絶対アドレス**としてロードされますので，二次マップにおけるメモリの割当てや配置に考慮が必要です．

SMLOAD ファイル・コール：モジュールを二次マップにローディングする．

MACSBUG↓入力：二次マップの "MACSbug" 状態に戻る．

UNLOAD コマンド：ロード・モジュールをライブラリのボリュームに格納．

OS↓入力："VERSAdos" 状態に戻る．

［注意］ SMLOAD が目的のタスクを二次マップにロードされるのに使用されているときは，一次マップへは **PR** コマンドを使用しても受け付けられない．

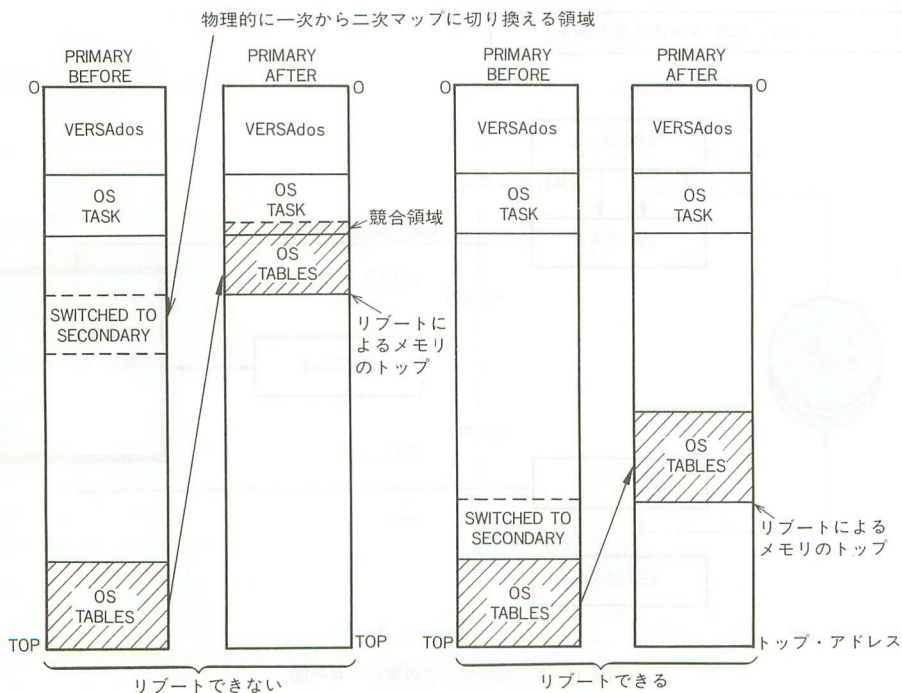



図 11・10 一次マップから二次マップへの切換え

11・2 RMS 68 K リアルタイム・モニタ

RMS 68 K/VDOS は，表 11・1 に示すようなプロセッサ管理，メモリ/タスク管理，ファイル管理，I/O 管理とからの 4 部構成になっており，マルチタスキング・ソフトウェアです．リアルタイムで動作する応用機器やマルチユーザで動作する応用機器やアセンブラ，リンカ，エディタ，パスカルなどのサービス・プログラムを利用する応用機器のソフトウェアの核となっています．

システム・ソフトウェアの構成は，図 11・11 に示すような構成になっており，ユーザはユーザ・プログラムを作成し，EXORmacs の **SYSGEN** ユーティリティを使用してシステム・ソフトウェアと結合することによって，応用機器のプログラムを完成することができます．

特長として次の項目があげられます．

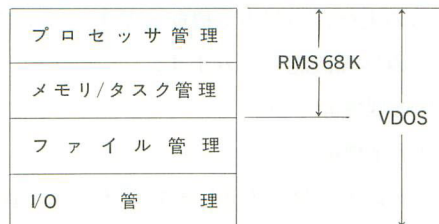
（1） リアルタイムのマルチタスク動作であり，タスク間の同期を行える．

（2） タスクの優先レベル（0～255）まで設定が可能であり，タスク数に制限を持たない．

（3） モニタ・タスク機能とサーバ・タスク機能を持つ．

（4） システム・ジェネレーションで固有のシステム構造が可能であり，ユーザの I/O デバイス・ドライバ機能記述が可能です．

（5） ソフトウェアとハードウェアの割込みに対するディスパッチング機能．

表 11・1 システム・ソフトウェア

リアルタイム・システムは，動作を逐次的に行なっていくバッチ・システムとは異なり，ある動作が終了しないうちに，ほかの動作を開始したり，続行したり終了したりすることができます．このようにリアルタイム・システムの並行処理機構は，あたかも，いくつかの動作が同時に実行されているかのように見えます．

RMS 68 K は，タスク制御機構，タスク間通信機構，メモリ管理機構，初期化機構の四つの機能より構成されています．

RMS 68 K は，図 11･12 に示すようにレベル 0～レベル 5 までの六つのレベル・モジュールから構成されており，レベル 3～5 は，タスクにより直接アクセスできるモジュールであり，レベル 0～2 は，レベル 3～5 のモジュールをサポートするモジュールです．

モジュールのメモリ必要量

RMS 68 K のメモリ構成は，レベル 0～4 の部分とレベル 5 の部分との二つの部分に分けることができます．

レベル 0～4：12 K バイトのプログラム・コード領域と 800 K バイトのデータ領域

レベル 5：4.5 K バイトのプログラム・コード領域

なお，RMS 68 K の供給は，5 枚の両面単密度ディスケットや 96 M バイト・ハード・ディスクなどにより供給されます．

レベル 0：プロセッサ管理機能

タスクへの実行権の割当て機能と基本同期機能と例外処理機能および割込み処理機能を含みます．

レベル 1：物理メモリ管理機能

レベル 2：ユーティリティ機能

レベル 3：タスクのアドレス空間とメモリ・セグメントの管理機能

レベル 4：タスクの生成，削除，実行制御機能

レベル 5：チャンネル用物理入出力機能（オプション）

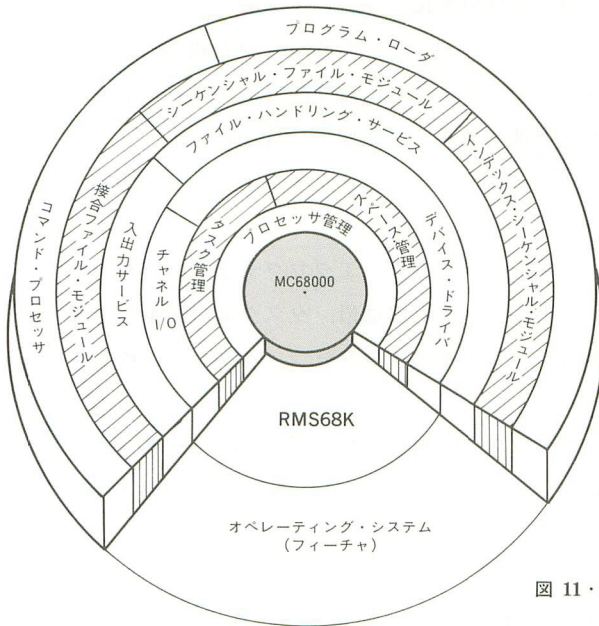



図 11・11 システム・ソフトウェアの構成

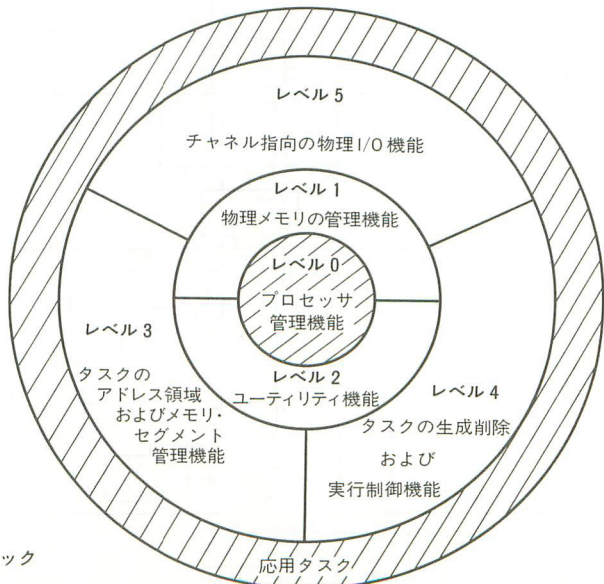



図 11・12 RMS 68 K の機能ブロック

応用システムのプログラムは，RMS 68 K のもとにタスクとして実行され，タスクは，タスクの優先度やスケジュールやタスクの要求資源の利用の可否に従ってスケジューリングされます．また，タスクは，機能的にまとまったある単位を実行するプログラムと，それに関連するデータ領域よりなります．

タスクのタイプとして，ユーザ・タスクとシステム・タスクの二つの基本的な型に分類でき，このいずれかに属します．

ユーザ・タスクは，通常の実用タスクであり，エグゼクティブ命令を用いて，メモリの割付け，解放，管理，他のタスクとの通信，タスク間の同期，タスクに従属する例外処理，物理入出力チャンネルのインターフェースなどの機能を実行することができます．

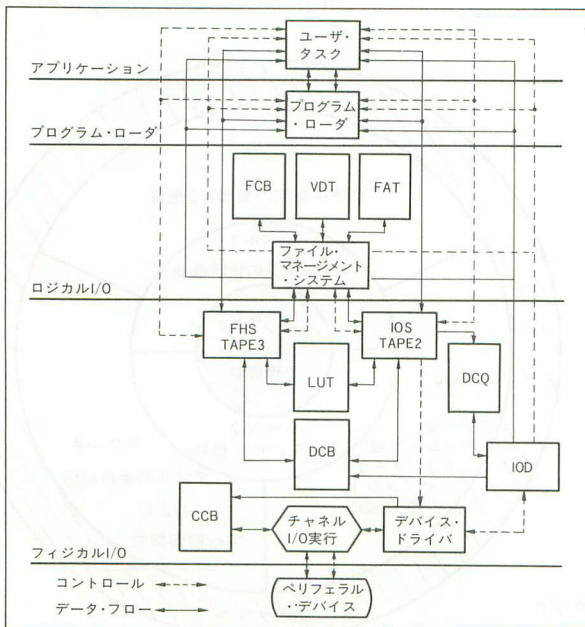
システム・タスクは，ユーザ・タスクに対してサービスを提供するために働くタスクであり，まったく制限なしに全資源をアクセスでき，すべてのエグゼクティブ命令を実行できます．

FCB：File Control Box
VDT：Volume Descriptor Table
FAT：File Access Table
FHS：File Handling Server
IOS：I/O Service
LUT：Logical Unit Table
DCQ：Device Connect Queue
DCB：Device Control Box
IOD：I/O Done
CCB：Channel Control Block
SVT：System Value Table

図 11・13　入/出力の要素

RMS 68 K は**タスク**により制御され，それぞれのタスクに対してタスク制御ブロック（TCB）を備え，タスク状態制御用リンク情報，各テーブルのリンク情報，レジスタ退避領域などにより成り立っています．

タスクは，図 11･14 に示される状態を遷移することにより，動的に実行されます．**セマフォ**は，タスク間の同期や複数のタスク間の共有データ領域の非同期な変更などに用います．また，セマフォの P/V オペレーションは，**WTSEM** と **SGSEM** の二つのディレクティブ命令によりサポートし，1，2，3 の 3 タイプのセマフォを持っています．

タイプ 1 は，複数のタスクによる一つの資源への排他的アクセスを実現します．

タイプ 2 は，タスクの順序処理を実現します．

タイプ 3 は，タイプ 1 の拡張であり，計数型のセマフォです．

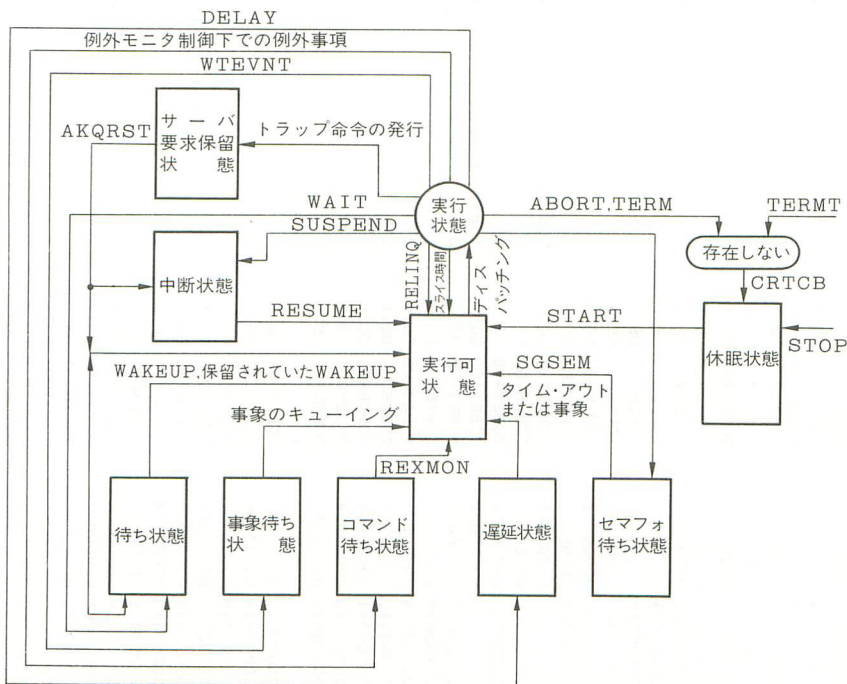



図 11・14　タスク状態遷移図

```
00000000 00000004      TCB      DS.L     1                 '!TCB'  DUMP EYE CATCHER
00000004 00000004      TCBALL   DS.L     1                 LINK FOR ALL TCB LIST
00000008 00000004      TCBGROUP DS.L     1                 LINK FOR TCB-THIS-GROUP LIST
0000000C 00000004      TCBREADY DS.L     1                 LINK FOR READY LIST
00000010 00000004      TCBNAME  DS.L     1                 4-BYTE TASK NAME
00000014 00000004      TCBSESSN DS.L     1                 SESSION CODE
00000018 00000008      TCBMON   DS.L     2                 MONITOR TASKNAME + SESSION CODE
00000020 00000004      TCBSEM   DS.L     1                 LINK TO NEXT SEMAPHORE WAITER
00000024 00000001      TCBCPRI  DS.B     1                 CURRENT TASK PRIORITY
00000025 00000001      TCBLPRI  DS.B     1                 TASK LIMIT PRIORITY
00000026 00000001      TCBRPRI  DS.B     1                 PRIORITY USED TO ENTER TASK IN READY LIST
00000027 00000001      TCBIOCNT DS.B     1                 COUNT OF PENDING INPUTS
00000028 00000002      TCBATTR  DS.W     1                 TASK ATTRIBUTES
         00000029      TCBATTI  EQU      TCBATTR+1         2ND BYTE OF ATTRIBUTES (INTERNAL FLAGS)
0000002A 00000002      TCBABORT DS.W     1                 ABORT CODE
0000002C 00000004      TCBSTATE DS.L     1                 CURRENT TASK STATE
         0000002D      TCBSTAT2 EQU      TCBSTATE+1        2ND BYTE OF STATE WORD
00000030 00000006      TCBTSTSM DS.W     3                 TST SEMAPHORE
00000036 00000004      TCBTST   DS.L     1                 POINTER TO TASK SEGMENT TABLE
0000003A 00000006      TCBASQSM DS.W     3                 ASQ SEMAPHORE
00000040 00000004      TCBASQ   DS.L     1                 POINTER TO ARQ
00000044 00000004      TCBCHAN  DS.L     1                 LINK TO NEXT CHANNEL CONTROL BLOCK
00000048 00000004      TCBEVECT DS.L     1                 ADDRESS OF TASK EXCEPTION VECTOR
0000004C 00000004      TCBTVECT DS.L     1                 ADDRESS OF TASK TRAP VECTOR
                       *
00000050 00000004      TCBTACC  DS.L     1                 ACCUMULATED TIME FOR THIS TASK (MSEC)
00000054 00000004      TCBTDLY  DS.L     1                 TIME TO RESTART JOB AFTER DELAY
00000058 00000004      TCBDLYLK DS.L     1                 LINK TO NEXT TCB IN DELAY LIST
0000005C 00000002      TCBUPDO  DS.W     1                 SAVE UPPER 1/2 OF DO ON TRAP 1'S
0000005E 00000002      TCBISRS  DS.W     1                 ISR ERROR CODE - SAVE FOR WAKEUP
00000060 00000004      TCBAINT  DS.L     1                 INTERVAL FOR PERIODIC ACTIVATION
00000064 00000004      TCBAASR  DS.L     1                 ADDR OF ASR FOR PERIODIC ACTIVATION
00000068 00000002      TCBAOPT  DS.W     1                 PERIODIC ACTIVATION OPTIONS
0000006A 00000002               DS.W     1
0000006C 00000004      TCBENTRY DS.L     1                 TASK INITIAL ENTRY POINT
00000070 00000002      TCBUSER  DS.W     1                 USER NUMBER ASSOCIATED WITH TASK
00000072 00000001      TCBSSP   DS.B     1                 EXEC STACK DEPTH
00000073 00000001      TCBUTRP  DS.B     1                 USER TRAP NUMBER - SET WHEN TRAP OCCURS
```

```
                   *
                   *       SAVE AREAS
                   *
00000074 00000020  TCBXREGS DR.L     8                  EXEC REGISTERS
00000094 00000004  TCBXA0   DS.L     1                  EXEX REGISTER A0
00000098 0000001B           DS.L     6
         000000B0  TCBSAFE  EQU      *                  EXEC REGISTERS
000000B0 00000004  TCBATSK  DS.L     1                  TASK NAME OF TASK THAT CAUSED TERMINATION
000000B4 00000004  TCBASES  DS.L     1                  SESSION OF TASK THAT CAUSED TERMINATION
000000B8 00000008  TCBBERR  DS.L     2                  INFO PLACED ON STACK BY BUS OR ADDRESS ERROR
                   *           ORG  $FA
000000C0 0000003A           DS.B     $FA-*
000000FA 00000002  TCBSR    DS.W     1                  USER'S STATUS REGISTER
         000000FB  TCBCC    EQU      TCBSR+1            USER'S CONDITION CODES
000000FC 00000004  TCBPC    DS.L     1                  USER'S PROGRAM COUNTER
         00000050  TCBROOM  EQU      $100-TCBSAFE
                   *
00000100 00000004  TCBD0    DS.L     1                  USER'S D0
         00000102  TCBRTCD  EQU      TCBD0+2
00000104 0000001C           DS.L     7                  USER'S D1 THRU D7
00000120 00000004  TCBA0    DS.L     1                  USER'S A0
00000124 00000014  TCBA1    DS.L     5                  USER'S A1 THRU A5
00000138 00000004  TCBA6    DS.L     1                  USER'S A6
0000013C 00000004  TCBUSP   DS.L     1                  USER'S A7
                   *
                   *        EXCEPTION MONITOR PARAMETERS
                   *
00000140 00000004  TCBEXM   DS.L     1                  EXCEPTION MONITOR TASK NAME
00000144 00000004  TCBEXMS  DS.L     1                  EXCEPTION MONITOR SESSION
00000148 00000004  TCBEMMSK DS.L     1                  EXCEPTION MONITOR MASK
0000014C 00000004  TCBEVMSK DS.L     1                  EXCEPTION MONITOR VALUE MASK
00000150 00000004  TCBEVLOC DS.L     1                  EXCEPTION MONITOR VALUE LOCATION
00000154 00000004  TCBEVALU DS.L     1                  EXCEPTION MONITOR VALUE
00000158 00000004  TCBECNT  DS.L     1                  EXCEPTION MONITOR MAX COUNT OF INSTRUCTIONS
0000015C 00000004           DS.L     1

         00000160  TSTBEGIN EQU      *                  OFFSET TO BEGINNING OF TST
```

図 11・15 TCB：タスク制御の構成例

```
0000000F     TSKASYST EQU     15                    SYSTEM TASK
0000000E     TSKAMRES EQU     14                    TASK IS MEMORY RESIDENT
0000000D     TSKACRIT EQU     13                    TASK IS CRITICAL TO OS - CRASH SYSTEM IF ABORTED
0000000B     TSKARELO EQU     11                    TASK IS RELOCATEABLE (NO MMU) - CONVERT ENTRY
00000007     TSKIUSEM EQU     7                     TASK HAS CREATED USER SEMAPHORE
00000006     TSKIEXM  EQU     6                     TASK CONTROLLED BY EXCEPTION MONITOR
00000005     TSKIEXMT EQU     5                     TASK IS EXCEPTION MONITOR FOR ANOTHER TASK
00000004     TSKIEVCT EQU     4                     TASK HAS OWN EXCEPTION VECTOR
00000003     TSKITVCT EQU     3                     TASK HAS OWN TRAP VECTOR
00000002     TSKILAST EQU     2                     TASK IS LAST TASK IN SESSION (SET ONLY BY TERM)
00000001     TSKIABRT EQU     1                     TASK WAS ABORTED
00000000     TSKIUVCT EQU     0                     TASK HAS CLAIMED USER VECTOR

                              *  タスク属性の定義

0000000F     TSKSDORM EQU     15                    TASK IS DORMANT
0000000E     TSKSBLCK EQU     14                    TASK IS BLOCKED
0000000D     TSKSSMWT EQU     13                    TASK IS BLOCKED ON EXEC SEMAPHORE
0000000C     TSKSEVWT EQU     12                    TASK IS WAITING FOR EVENT
0000000B     TSKSAKWT EQU     11                    TASK IS WAITING FOR SERVICE REQ ACKNOWLEDGEMENT
0000000A     TSKSWTEM EQU     10                    TASK IS WAITING FOR COMMAND FROM EXMON
00000009     TSKSSUSP EQU     9                     TASK IS SUSPENDED
             *
00000007     TSK2TRMP EQU     7                     TASK HAS PENDING TERMINATION
00000006     TSK2RTEX EQU     6                     TASK WILL RETURN TO EXEC
00000005     TSK2EVWK EQU     5                     TASK IS HEADED FOR ASR
00000004     TSK2NRDY EQU     4                     TASK IS ON READY LIST
00000003     TSK2WKWT EQU     3                     TASK HAS PENDING WAKEUP
00000002     TSK2ACK2 EQU     2                     TERM MESSAGE TO SERVER SENT WHILE ACK OUTSTANDING

                              *  タスク状態の定義

00000001     TEMTRAP1 EQU     1                     TRAP 1 IS MONITORED
00000010     TEMBUSER EQU     16                    BUS ERROR IS MONITORED
00000011     TEMADDER EQU     17                    ADDRESS ERROR IS MONITORED
00000012     TEMILLEG EQU     18                    ILLEGAL INSTRUCTION IS MONITORED
00000013     TEMZDIV  EQU     19                    ZERO DIVIDE IS MONITORED
00000014     TEMCHK   EQU     20                    CHK INSTRUCTION IS MONITORED
00000015     TEMTRAPV EQU     21                    TRAPV INSTRUCTION IS MONITORED
00000016     TEMPRIV  EQU     22                    PRIVILEGE VIOLATION IS MONITORED
00000017     TEML1010 EQU     23                    LINE 1010 IS MONITORED
00000018     TEML1111 EQU     24                    LINE 1111 IS MONITORED
             *
```

```
0000001B      TEMMCNT  EQU      27                    MAX INSTRUCTION COUNT SUPPLIED
0000001C      TEMTRAC  EQU      28                    SINGLE INSTRUCTION TRACE REQUESTED (IF VCHG=0)
0000001C      TEMVEQU  EQU      28                    VALUE = TEST REQUESTED (IF VCHG=1)
0000001D      TEMVCHG  EQU      29                    VALUE CHANGE TEST REQUESTED

                             *  タスク・マスクの定義
```

```
00000000 00000004    TST      DS.L    1                  !TST DEBUG TOOL
00000004 00000001    TSTNSEGS DS.B    1                  NUMBER OF SEGMENTS
00000005 00000001    TSTCSEGS DS.B    1                  NUMBER OF SEGMENTS CURRENTLY EXISTING
00000006 00000002    TSTLMMU  DS.W    1                  INDEX TO LAST MMU LOAD LINE
00000008 00000002    TSTLATTR DS.W    1                  INDEX TO LAST SEGMENT ATTRIBUTE LINE
0000000A 00000002    TSTSTAT  DS.W    1                  MMU STATUS
0000000C 00000020    TSTMMU   DS.W    4*SEGMENTS         MMU LOAD INFORMATION
0000002C 00000020    TSTATTR  DS.W    4*SEGMENTS         SEGMENT ATTRIBUTE BLOCK
0000004C 00000004    TSTWTSK  DS.L    1                  NAME OF TASK WITH WINDOW TO ADDRESS SPACE
         00000050    TSTLEN   EQU     *                  LENGTH OF TASK SEG TABLE
                     *
                             *  TSTの構成

         0000000F    SEGAUSED EQU     15                 SEGMENT ENTRY IN USE
         0000000E    SEGAROM  EQU     14                 SEGMENT IS READ ONLY
         0000000D    SEGASHR  EQU     13                 SEGMENT IS SHARED WITHIN SESSION
         0000000C    SEGAGLBL EQU     12                 SEGMFNT IS GLOBAL
         0000000B    SEGAMMID EQU     11                 SEGMENT IS MEMORY MAPPED IO SPACE
         0000000A    SEGAPROM EQU     10                 SEGMENT IS PHYSICAL ROM

                        *  セグメント属性ビットの定義

         00000000    TSTLB    EQU     0                  LOGICAL BEGIN
         00000002    TSTLE    EQU     2                  LOGICAL END
         00000004    TSTPO    EQU     4                  PHYSICAL OFFSET
         00000006    TSTCTL   EQU     6                  CONTROL (NULL BYTE + CONTROL BYTE)

                         *  MMU制御レジスタ値
```

図 11・16　TST：タスク・セグメント・テーブルの例

タイプ1セマフォ例

時間→ タスクA	タスクB	タスクC	セマフォBUFRDY 初期カウント			セマフォMSGSNT 初期カウント		
			初期カウント	信号カウント	FIFO	初期カウント	信号カウント	FIFO
	ATSEM(BUFRDY)				空である			空である
CRSEM(BUFRDY)			5	5				
CRSEM(MSGSNT)						0	0	
WTSEM(MSGCNT)							−1	タスクA
	ATSEM(MSGCNT)							
	WTSEM(BUFRDY)			4				
	(メッセージをバッファに入れる)							
	SGSEM(MSGSNT)						0	空となる
(メッセージを処理する)								
SGSEM(BUFRDY)				5				
WTSEM(MSGSNT)							−1	タスクA
		ATSEM(BUFRDY)						
		ATSEM(MSGSNT)						
		WTSEM(BUFRDY)		4				
		(メッセージをバッファに入れる)						
		SGSEM(MSGSNT)					0	
		WTSEM(BUFRDY)		3				
		(メッセージをバッファに入れる)						
		SGSEM(MSGSNT)					1	空となる
(メッセージを処理する)								

タイプ2セマフォ例

時間→ タスクA	タスクB	タスクC	R1SEM		
			初期カウント	信号カウント	FIFO
	ATSEM			1	空である
	WTSEM			0	
ATSEM					

WTSEM			タイプ1では使用しない	−1	タスクAがセマフォ・キューに登録される
		ATSEM WTSEM		−2	セマフォ・キューにさらにタスクCが追加される
	（資源を使用する） SGSEM			−1	タスクAがセマフォ・キューから削除され，実行可リストに移される
（資源を使用する） SGSEM				0	タスクCがセマフォ・キューから削除され，実行可リストに移される
		（資源を使用する） SGSEM		1	

タイプ3セマフォ例

時間↓	タスクA	タスクB	タスクC	セマフォR2ASEM			セマフォR2BSEM		
				初期カウント	信号カウント	FIFO	初期カウント	信号カウント	FIFO
		ATSEM(A) CRSEM(B) WTSEM(A)			0 −1	空である タスクB	0	0	空である
	CRSEM(A) （処理を行う） SGSEM(A) DESEM(A)				0				
この時点でR2ASEM⇨は削除される		DESEM(A) （処理を行う） SGSEM(B) DESEM(B)				空となる		1	
この時点でR2BSEM⇨は削除される			ATSEM(B) WTSEM(B) DESEM(B) （処理を行う）					0	

図 11・17 セマフォの例

```
   LOC     OBJ          SOURCE                    < Assembler for MC68k on CP/M >

                     ;;;;;;;;;;;;;;;;;;;;;;;;;
                     ;                       ;
                     ;    MONITOR PROGRAM    ;
                     ;                       ;
                     ;;;;;;;;;;;;;;;;;;;;;;;;;
            =000F ACIAC     EQU     0F8001H     ; ACIA CONTROL REGISTER
            8001
            =000F ACIAD     EQU     0F8003H     ; ACIA DATA REGISTER
            8003
            =0000 RECREADY  EQU     0           ; RECEIVE READY FLAG
            0000
            =0000 SENDREADY EQU     1           ; SEND READY FLAG
            0001
            =0000 SENDOVER  EQU     3           ; SEND OVER FLAG
            0003
            =0000 ESCAPE    EQU     82H         ; ESCAPE CHARACTER WITH PARITY
            0082
            =0002 ENDOERAM  EQU     20000H      ; END OF RAM
            0000
            =0001 SSPBOT    EQU     10000H      ; SYSTEM STACK BOTTOM
            0000
            =0000 CONTAD    EQU     10000H-4    ; CONTINUATION ADDRESS
            FFFC
            =0000 STATUS    EQU     10000H-6    ; USER STATUS SAVE AREA ADDRESS
            FFFA
            =0000 UREGSV    EQU     10000H-70   ; TOP OF STACK WHEN USER REG'S ARE SAVED
            FFBA
            =0000 CLEAR     EQU     0           ; ZERO NULL NIL NASHI
            0000
            =0000 BS        EQU     7FH         ; BACK SPACE
            007F
            =0000 CAN       EQU     18H         ; CANCEL
            0018
            =0000 CR        EQU     0DH         ; RETURN
            0000
            =0000 LF        EQU     0AH         ; LINE FEED (NEGLECT)
            000A
            =0000 DUMMYVAL  EQU     0           ; DUMMY VALUE
            0000

0000 0000 =0000              ORG     0H
          0000
0000 0000 0001              DEFL    SSPBOT             ;INITIAL SSP
     0002 0000
0000 0004 0000              DEFL    MONITSTART         ;INITIAL PC
     0006 0500
0000 0008 0000              DEFL      ERROR            ;BUS ERROR
     000A 0300
0000 000C 0000              DEFL     ADDRERR           ;ADDRESS ERROR
     000E 0278
0000 0010 0000              DEFL     LLLDGL            ;ILLDGAL INSTRUCTION
     0012 0322
0000 0014 0000              DEFL      ZERO             ;ZERO DIVIDE
     0016 0312
0000 0018 0000              DEFL      ERROR            ;CHK INSTRUCTION
     001A 0300
0000 001C 0000              DEFL      ERROR            ;TRAPV INSTRUCTION
     001E 0300
0000 0020 0000              DEFL    PRIVERR            ;PRIVILPGE VIOLATION
     0022 0332
0000 0024 0000              DEFL    TRACE              ;TRACE
     0026 0356
0000 0028 0000              DEFL      ERROR            ;LINE 1010 EMURATOR
     002A 0300
0000 002C 0000              DEFL      ERROR            ;LINE 1111 EMULATOR
     002E 0300
0000 0030 =0000              ORG     60H
          0060

0000 0060 0000              DEFL      ERROR            ;SPURIOUS INTERRUPT
     0062 0300
0000 0064 0000              DEFL      ERROR            ;AUTOVECTOR 1
     0066 0300
```

12. MC68000の将来の発展方向

マイクロプロセッサは，チップ寸法の制約，プロセス技術の制約，性能の要求など多くの制約の中でおのおのの機構を取り入れ，よりいっそう強化する方向に向かっております．アーキテクチャやソフトウェア面も，より強力なモニタが作成され，その管理下のもとにシステムが運用されるようになります．MOS技術の方向，ソフトウェアやハードウェア・コストの傾向と68000の発展方向を説明しています．

12・1 MOS技術の方向

マイクロプロセッサの生産におけるMOS技術の進歩を図12・1に示します．1973年の8ビットMPUスタート時の**NMOS**技術と現在の32ビットMPUの**HMOS**技術との間に容量/チップを例にしても約10倍以上と数段の躍進がなされたことがおわかりいただけるでしょう．

図12・2は，年々の技術躍進をなしえたバックグラウンド資料として，ダイ・サイズの傾向を載せておきました．ダイ・サイズにおいても8ビットどきと比較して3倍以上の技術進歩がなされています．

種類＼項目	ゲート・レングス	エリア・ファクタ	ゲート・ディレイ（TIME t_D）	容量/チップ
標 準 NMOS	6. MICRONS	1	10 ns	50 k
HMOS I	3.5 MICRONS	.48	4～5 ns	100 k
HMOS II	2.5 MICRONS	.26	2～3 ns	250 k
HMOS III	1.7 MICRONS	.15	1 ns	650 k

（年 ↓）

図 12・1 MOSテクノロジーとパフォーマンスの発展

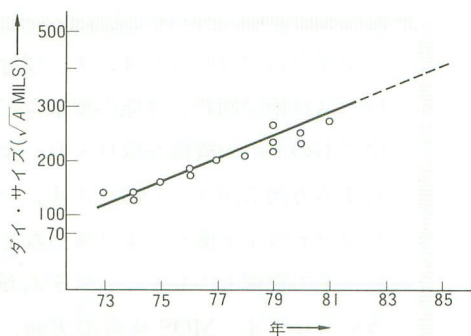



図 12・2 マイクロプロセッサのダイ・サイズと傾向

12・2 ソフトとハード・コストの方向

ソフトウェア・コストとハードウェア・コストは，図12・3に示すように**反比例**の関係になりつつあり，年々ハードウェア・コストは減少し，逆にソフトウェア・コストは増加しています．このことは，プロセス技術と生産設備技術の著しい躍進があげられます．この傾向は，今後なおいっそう増幅されてゆくと思われます．図12・4はシステム・コストの傾向を示し，ハードウェアとソフトウェアの分岐が1985年ごろに完全に逆転し，ソフトウェア・コストがハードウェア・コストを大幅に上回ってくることです．

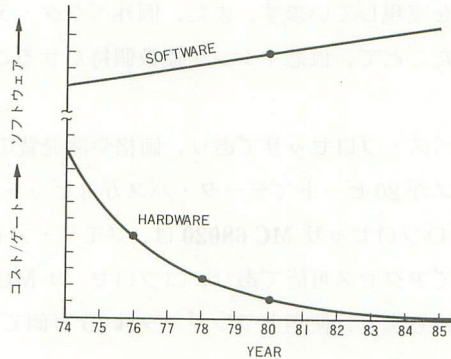



図 12・3 ソフトウェアとハードウェア・コストの推移

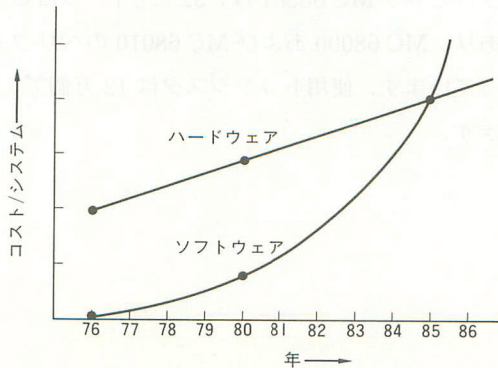



図 12・4 ハードウェアとソフトウェアのシステム・コスト

12・3　MC 68000 の発展方向

モトローラのマイクロプロセッサの**設計サイクル**を見ますと，図 12・5 に示すように 5 ～ 6 年をサイクルに展開されています．1973 から 1979 年までは 8 ビット系の確立に，1979から 1985 年までは 16 ビットの確立に， 32 ビットの確立には 1987 年ごろまで費やされるでしょう．

仮想マシン（VM）**プロセッサ MC 68010** は，MC 68000 では実メモリ空間に命令がないときにホルトが起きても，その命令をアボードしてリランする機能がなかったので，アボード機能を追加し，例外ベクタ・テーブルを再配置することによって仮想メモリを実現しています．また，例外ベクタ・テーブルを動的に変更できるようになったことで，仮想マシンを複数個持たせることができるようになりました．

MC 68008 は縮小バス・プロセッサであり，価格や開発費の低減を目的としており，アドレス・バスが 20 ビットでデータ・バスが 8 ビットです．

32 ビット・マイクロプロセッサ **MC 68020** は，メモリ・アドレスが 16 M バイトから **4 G** バイトまでアクセス可能であり，コプロセッサ MC 68881 に接続できるアーキテクチャを持ちます．使用トランジスタは 15 万個で，HMOS Ⅲ のプロセス技術で作られています．

浮動小数点コ・プロセッサ MC 68881 は，32 ビット・プロセッサ MC 68020 用結合プロセッサであり，MC 68000 および MC 68010 のペリフェラルとしても使用できるようになっています．使用トランジスタは 12 万個で，HMOS Ⅲ のプロセス技術で作られます．

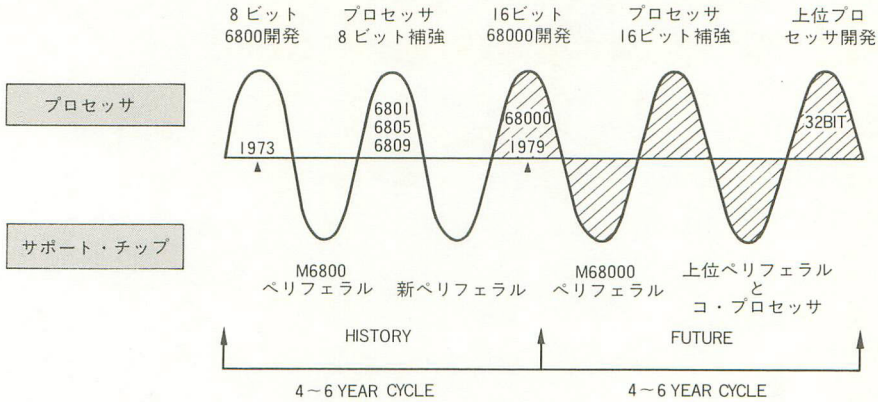



図 12・5 モトローラ生産開発プラン

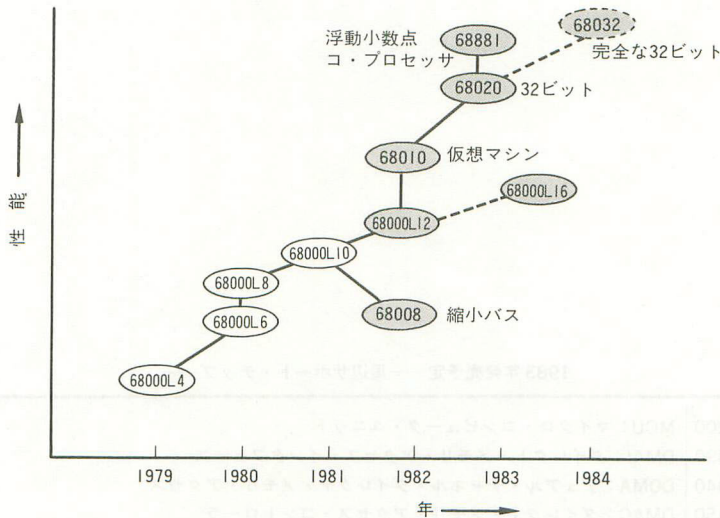
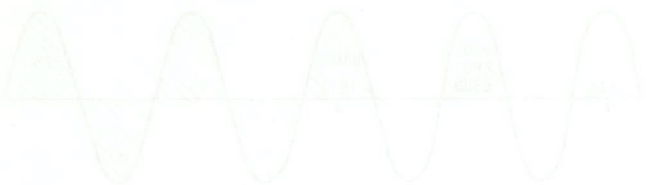



図 12・6 MC 68000ファミリの開発予定

1983年発売予定——周辺サポート・チップ——

MC 68200	MCU：マイクロ・コンピュータ・ユニット
MC 68430	DMAI：ダイレクト・メモリ・アクセス・インタフェース
MC 68440	DOMA：デュアル・チャネル・ダイレクト・メモリ・アクセス
MC 68450	DMAC：ダイレクト・メモリ・アクセス・コントローラ
MC 68452	BAM：バス・アービトレーション・モジュール
MC 68454	IMDC：インテリジェント・マルチディスク・コントローラ
MC 68459	DPLL：ディスク・フェーズ・ロック・ループ
MC 68562	DUSCC：デュアル・ユニバーサル・シンクロナス・コミュニケーション・コントローラ
MC 68564	SIO：シリアルI/O
MC 68590	LANCE：ローカル・エリア・ネットワーク・コントローラ
MC 68901	MFP：マルチ・ファンクション・ペリフェラル

付　　録

付録 1.　レジスタ転送言語定義（RTL：Register Transfer Language）

以下にレジスタ転送言語定義で命令セットの詳細な動作説明をします.

オペランド：

- **An**：アドレス・レジスタ
- **Dn**：データレジスタ
- **Rn**：データ・レジスタまたはアドレス・レジスタのいずれか
- **PC**：プログラム・カウンタ
- **SR**：ステータス・レジスタ
- **CCR**：コンディション・コード（SRの下位バイト）
- **SSP**：スーパバイザ・スタック・ポインタ
- **USP**：ユーザ・スタック・ポインタ
- **SP**：アクティブ・スタック・ポインタ（Ａ７）と同じ
- **X**：拡張オペランド（コンディション・コードからの）
- **Z**：ゼロ・コンディション・コード
- **V**：オーバフロー・コンディション・コード
- **Immediate Data**：イミディエート・データ・オペランド
- **d**：アドレス変位（displacement）
- **Source**：ソース位置の指定
- **Destination**：ディスティネーション位置の指定
- **Vector**：エクセプション・ベクタの位置

サブフィールドと修飾記号（qualifiers）

- **＜ビット＞OF＜オペランド＞**　オペランドのある一つのビットを選ぶ.
- **＜オペランド＞〔＜ビット番号＞：＜ビット番号＞〕**　オペランドのサブフィールドを選ぶ.
- **（＜オペランド＞）**　参照されたオペランドの中身
- **＜オペランド＞**$_{10}$　オペランドはBCD（2進化10進）であり，オペレーションは10進で実行される.
- **＜オペランド＞@＜モード＞**　オペランドで指定したアドレス・レジスタがオペランド・データのメモリ位置を指示することを表すレジスタ間接オペレータ・オプションのモード修飾記号は，－，＋，（ｄ），および（ｄ，ｉｘ）.

オペレーション：オペレーションは，2オペランド操作，単一オペランド操作およびその他に分けられる.

2オペランド操作：これらの命令は＜オペランド＞＜オペレーション＞＜オペランド＞で表され，ここでの＜オペレーション＞は次のいずれかである.

- →　左のオペランドが右のオペランドで指定される位置に移される.
- ↔　二つのオペランド内容が交換される.

+　両オペランドが加算される．

-　右側のオペランドが左側のオペランドから減算される．

*　左右のオペランドが乗算される．

/　左のオペランドが右のオペランドで除算される．

∧　左右オペランドが論理ANDされる．

∨　左右オペランドが論理ORされる．

⊕　左右オペランドが論理EORされる．

<　関係テスト，左のオペランドが右のオペランドより小なら真．

>　関係テスト，左のオペランドが右のオペランドより大なら真．

≠　関係テスト，左のオペランドが右のオペランドに等しくなければ真．

shifted by ⎫
rotated by ⎭ 左のオペランドが，右のオペランドで規定される位置数だけシフト，またはローテートされる．

単一オペランド：

~<**オペランド**>　オペランドの論理的補数をとる．

<**オペランド**>**Sign-extended**　オペランドが符号拡張される．上位半分の全ビットが下位半分の最上位ビットと等しくされる．

<**オペランド**>**tested**　オペランドが0と比較され，その結果がコンディション・コードのセットに使用される．

その他：

TRAP　PC→SSP@-；SR→SSP@-；(ベクタ)→PCSと等価．

STOP　ストップ状態に入り，割込みを待つ．

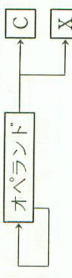
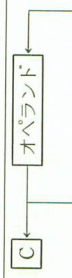
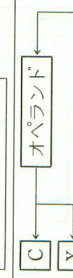
If<**コンディション**>**then**<**オペレーション**>**else**<**オペレーション**>

そのコンディションでテストされる．もし真ならthenの次の操作が実行される．もしコンディションが偽でオプションのelse部があればelseの操作が実行される．もしコンディションが偽でオプションのelse部がなければ，命令は操作されない．

付録 2.　MC 68000 アセンブラ命令一覧表

（1）

分類	命令	B	W	L	オペランド	モード	操作	X	N	Z	V	C	備考
データ転送命令	MOVE	●	●	●	<ea>,<ea>	→	(soc)→des	—	＊	＊	0	0	モード {des→DA, soc→AL, (Byte→Anモード不可)}
	MOVEA		●	●	<ea>,An	AL		—	—	—	—	—	
	MOVEP		●	●	Dx,d(Ay)	—		—	—	—	—	—	
			●	●	d(Ay),Dx	—		—	—	—	—	—	
	MOVEQ			●	#<data>,Dn	—		—	＊	＊	0	0	データ範囲(−$80～$7F)
	MOVEM		●	●	<reg-list>,<ea>	CA		—	—	—	—	—	(An)@−モード可
			●	●	<ea>,<reg-list>	C		—	—	—	—	—	(An)@+モード可)
	EXG			●	Rx,Ry	—	Rx←→Ry	—	—	—	—	—	Dn←→Anに限り
	LEA			●	<ea>,An	C	(soc)→An	—	—	—	—	—	EXG,Dn,An
	SWAP		●		Dn	—	REG[31:16]←→REG[15:0]	—	＊	＊	0	0	
	PEA			●	<ea>	C	des→SP@−	—	—	—	—	—	
	LINK	—	—	—	An,#<disp>	—	An→SP@−, SP→An, SP+d→SP	—	—	—	—	—	
	UNLK	—	—	—	An	—	An→SP, SP@+→An	—	—	—	—	—	
演算命令	ADD	●	●	●	<ea>,Dn	AL	(des)+(soc)→des	＊	＊	＊	＊	＊	Byte→Anモード不可
		●	●	●	Dn,<ea>	AM		＊	＊	＊	＊	＊	
	ADDA		●	●	<ea>,An	AL		—	—	—	—	—	
	ADDI	●	●	●	#<data>,<ea>	DA	(des)+ID→des	＊	＊	＊	＊	＊	
	ADDQ	●	●	●	#<data>,<ea>	A	(des)+ID(0～7)→des	＊	＊	＊	＊	＊	Byte→Anモード不可
	ADDX	●	●	●	Dy,Dx	—	(des)+(soc)+X→des	＊	＊	＊	＊	＊	
		●	●	●	-(Ay),-(Ax)	—		＊	＊	＊	＊	＊	
	SUB	●	●	●	<ea>,Dn	AL	(des)−(soc)→des	＊	＊	＊	＊	＊	Byte→Anモード負可
		●	●	●	Dn,<ea>	AM		＊	＊	＊	＊	＊	
	SUBA		●	●	<ea>,An	AL		—	—	—	—	—	

分類	命令	オペランド・サイズ B	W	L	オペランド	モード	操作	フラグ X	N	Z	V	C	備考
演算命令	SUBI	●	●	●	#〈data〉,〈ea〉	DA	(des)－ID→des	*	*	*	*	*	
	SUBQ	●	●	●	#〈data〉,〈ea〉	A	(des)－ID(0～7)→des	*	*	*	*	*	Byte→Anモード不可
	SUBX	●	●	●	Dy,Dx	—	(des)－(soc)－X→des	*	*	*	*	*	
		●	●	●	-(Ay),-(Ax)	—		*	*	*	*	*	
	MULS		●		〈ea〉,Dn	D	(des)*(soc)→des	—	*	*	0	0	符号付き
	MULU		●			D		—	*	*	0	0	符号なし
	DIVS		●			D	(des)/(soc)→des	—	*	*	*	0	符号付き
	DIVU		●			D		—	*	*	*	0	符号なし
	CLR	●	●	●	〈ea〉	DA	0→des	—	0	1	0	0	
	NEG	●	●	●		DA	0－(des)→des	*	*	*	*	*	
	NEGX	●	●	●		DA	0－(des)－X→des	*	*	*	*	*	
	TAS	●				DA	(des) tested→CC 1→bit 7 of des	—	*	*	0	0	
	TST	●	●	●		DA	(des) tested→CC	—	*	*	0	0	データ0と比較
	EXT		●	●	Dn	—	(des)符号拡張→des	—	*	*	0	0	
	CMP	●	●	●	〈ea〉,Dn	AL	(des)－(soc)	—	*	*	*	*	Byte→Anモード不可
	CMPA		●	●	〈ea〉,An	AL		—	*	*	*	*	
	CMPM	●	●	●	(Ay)+,(Ax)+	—		—	*	*	*	*	
	CMPI	●	●	●	#〈data〉,〈ea〉	DA		—	*	*	*	*	
論理命令	AND	●	●	●	〈ea〉,Dn	D	(des)∧(soc)→des	—	*	*	0	0	
		●	●	●	Dn,〈ea〉	AM		—	*	*	0	0	
	ANDI	●	●	●	#〈data〉,〈ea〉	DA	(des)∧ID→des	—	*	*	0	0	SRモード可
	OR	●	●	●	〈ea〉,Dn	D	(des)∨(soc)→des	—	*	*	0	0	
		●	●	●	Dn,〈ea〉	AM		—	*	*	0	0	
	ORI	●	●	●	#〈data〉,〈ea〉	DA	(des)∨ID→des	—	*	*	0	0	SRモード可

（2）

分類	命令	オペランド・サイズ B	W	L	オペランド	モード	操作	フラグ X	N	Z	V	C	備考
論理命令	EOR	●	●	●	Dn,<ea>	DA	(des)⊕(soc)→des	—	*	*	0	0	
	EORI	●	●	●	#<data>,<ea>	DA	(des)⊕ID→des	—	*	*	0	0	SRモード可
	NOT	●	●	●	<ea>	DA	~(des)→des	—	*	*	0	0	
シフト命令	ASL	●	●	●	Dx,Dy	—	C, X ← オペランド ← 0	*	*	*	*	*	
		●	●	●	#<data>,Dy	—		*	*	*	*	*	
			●		<ea>	AM		*	*	*	*	*	
	ASR	●	●	●	Dx,Dy	—	オペランド（最上位ビット保持）→ C, X	*	*	*	*	*	
		●	●	●	#<data>,Dy	—		*	*	*	*	*	
			●		<ea>	AM		*	*	*	*	*	
	LSL	●	●	●	Dx,Dy	—	C, X ← オペランド ← 0	*	*	*	0	*	
		●	●	●	#<data>,Dy	—		*	*	*	0	*	
			●		<ea>	AM		*	*	*	0	*	
	LSR	●	●	●	Dx,Dy	—	0 → オペランド → C, X	*	*	*	0	*	
		●	●	●	#<data>,Dy	—		*	*	*	0	*	メモリ・シフトは1ビットのみシフト
			●		<ea>	AM		*	*	*	0	*	
ローテーション命令	ROL	●	●	●	Dx,Dy	—	C ← オペランド（循環）	—	*	*	0	*	
		●	●	●	#<data>,Dy	—		—	*	*	0	*	
			●		<ea>	AM		—	*	*	0	*	
	ROR	●	●	●	Dx,Dy	—	オペランド（循環）→ C	—	*	*	0	*	
		●	●	●	#<data>,Dy	—		—	*	*	0	*	
			●		<ea>	AM		—	*	*	0	*	
	RJXL	●	●	●	Dx,Dy	—	C, X ← オペランド ← X（循環）	*	*	*	0	*	
		●	●	●	#<data>,Dy	—		*	*	*	0	*	
			●		<ea>	AM		*	*	*	0	*	

分類	命令	オペランド・サイズ B	W	L	オペランド	モード	操作	フラグ X	N	Z	V	C	備考
ローテーション命令	ROXR	●	●	●	Dx,Dy	—	オペランド→C, X	*	*	*	0	*	メモリ・ローテーションは1ビットのみシフト
		●	●	●	#<data>,Dy	—		*	*	*	0	*	
			●		<ea>	AM		*	*	*	0	*	
ビット操作命令	BTST	●		●	Dn,<ea>	D	~(指定ビット)→Z	—	—	*	—	—	IDモード不可↓
		●		●	#<data>,<ea>	D		—	—	*	—	—	2ワード目にビット番号
	BSET	●		●	Dn,<ea>	DA	~(指定ビット)→Z	—	—	*	—	—	
		●		●	#<data>,<ea>	DA	1→(指定ビット)	—	—	*	—	—	2ワード目にビット番号
	BCLR	●		●	Dn,<ea>	DA	~(指定ビット)→Z	—	—	*	—	—	
		●		●	#<data>,<ea>	DA	0→(指定ビット)	—	—	*	—	—	2ワード目にビット番号
	BCHG	●		●	Dn,<ea>	DA	~(指定ビット)→Z	—	—	*	—	—	
		●		●	#<data>,<ea>	DA	~(指定ビット)→指定ビット	—	—	*	—	—	2ワード目にビット番号
BCD命令	ABCD	●			Dy,Dx	—	$(des)_{10}+(soc)_{10}+X \rightarrow des$	*	u	*	u	*	
		●			-(Ay),-(Ax)	—		*	u	*	u	*	
	SBCD	●			Dy,Dx	—	$(des)_{10}-(soc)_{10}-X \rightarrow des$	*	u	*	u	*	
		●			-(Ay),-(Ax)	—		*	u	*	u	*	
	NBCD	●			<ea>	DA	$0-(des)_{10}-X \rightarrow des$	*	u	*	u	*	
プログラム・コントロール命令	Bcc	●	●		<label>	—	if cc then PC+d→PC	—	—	—	—	—	PC@(d)モードのみ可　符号付き8 or 16ビット
	DBcc		●		Dn,<label>	—	if~cc then Dn−1→Dn if Dn≠−1 then PC+d→PC	—	—	—	—	—	PC@(d)モードのみ可　符号付き16ビット
	Scc	●			<ea>	DA	if cc then I's→des else 0's→des	—	—	—	—	—	
	BRA	●	●		<ladel>	—	PC+d→PC	—	—	—	—	—	PC@(d)モードのみ可　符号付き8 or 16ビット
	BSR	●	●		<label>	—	PC→SP@−, PC+d→PC	—	—	—	—	—	

（3）

分類	命令	オペランド・サイズ B	W	L	オペランド	モード	操作	フラグ X	N	Z	V	C	備考
プログラム・コントロール命令	JMP	—	—	—	〈ea〉	C	(des)→PC	—	—	—	—	—	
	JSR	—	—	—	〈ea〉	C	PC→SP@−, (des)→PC	—	—	—	—	—	
	RTR	—	—	—		—	SP@+→CC, SP@+→PC	＊	＊	＊	＊	＊	
	RTS	—	—	—		—	SP@+→PC	—	—	—	—	—	
システム・コントロール命令	RESET	—	—	—		—	外部デバイスのリセット	—	—	—	—	—	特権命令
	RTE	—	—	—		—	SP@+→SR, SP@+→PC	＊	＊	＊	＊	＊	
	STOP	—	—	—	#×××	—	ID→SR, stop	＊	＊	＊	＊	＊	
	MOVE			●	USP,An	—	(USP)→An	—	—	—	—	—	
				●	An,USP	—	An→USP	—	—	—	—	—	
			●		〈ea〉,SR	D	(soc)→SR	＊	＊	＊	＊	＊	
	ANDI		●		#〈data〉,SR	—	(SR)∧ID→SR	—	＊	＊	0	0	
	ORI		●			—	(SR)∨ID→SR	—	＊	＊	0	0	
	EORI		●			—	(SR)⊕ID→SR	—	＊	＊	0	0	
	TRAP	—	—	—	#〈vector〉	—	PC→SSP@−, SR→SSP@− vector→PC	—	—	—	—	—	トラップ発生命令
	TRAPV	—	—	—		—	if V then TRAP	—	—	—	—	—	
	CHK		●		〈ea〉,Dn	D	if Dn<0 or Dn>(soc) then trap	—	＊	u	u	u	
	ANDI	●			#〈data〉,CCR		(CCR)∧ID→CCR	—	＊	＊	0	0	ステータス・レジスタ変更命令
	ORI	●					(CCR)∨ID→CCR	—	＊	＊	0	0	
	EORI	●					(CCR)⊕ID→CCR	—	＊	＊	0	0	
	MOVE		●		〈ea〉,CCR	D	(soc)→CCR	＊	＊	＊	＊	＊	
			●		SR,〈ea〉	DA	SR→(des)	—	—	—	—	—	
その他	NOP	—	—	—				—	—	—	—	—	何の処理もしない

（4）

実効アドレス・モード <ea>

（×：指定可）

実効アドレス・モード	FA MOD	FA REG	D	M	C	A	DA	CA	AM	AL	<ea>	ディスプレースメント
Dn	000	番号	×			×	×			×	Dn	—
An	001	番号				×				×	An	—
An@	010	番号	×	×	×	×	×	×	×	×	(An)	—
An@+	011	番号	×	×		×	×		×	×	(An)+	—
An@−	100	番号	×	×		×	×		×	×	-(An)	—
An@(d)	101	番号	×	×	×	×	×	×	×	×	d(An)	符号付き16ビット
An@(d, ix)	110	番号	×	×	×	×	×	×	×	×	d(An,Rn,W) d(An,Rn,L)	符号付き8ビット
xxx, W	111	000	×	×	×	×	×	×	×	×	xxx	—
xxx, L	111	001	×	×	×	×	×	×	×	×	xxxxxx	—
PC@(d)	111	010	×	×	×					×	*+<ab,sym> *-<ab,sym> <rel.sym> <ab.sym>	符号付き16ビット
PC@(d, ix)	111	011	×	×	×					×	<rel.sym>(Rn.W) <rel.sym>(Rn.L)	符号付き8ビット
#xxx	111	100	×	×						×	#xxx	—
SR	111	100									SR,CCR	特殊モード

プログラム・コントロール命令でのコンディション（cc）

ニモニック	コンディション	エンコード	テスト条件	
T	true	0000	1	
F	false	0001	0	
HI	high	0010	$\overline{C}\cdot\overline{Z}$	
LS	low or same	0011	$C+Z$	
CC	carry clear	0100	$\overline{C}$	(High or Same)
CS	carry set	0101	C	(Low)
NE	not equal	0110	$\overline{Z}$	
EQ	equal	0111	Z	
VC	overflow clear	1000	$\overline{V}$	
VS	overflow set	1001	V	
PL	plus	1010	$\overline{N}$	
MI	minus	1011	N	
GE	greater or equal	1100	$N\cdot V+\overline{N}\cdot\overline{V}$	符号ビット考慮される
LT	less then	1101	$N\cdot\overline{V}+\overline{N}\cdot V$	符号ビット考慮される
GT	greater than	1110	$N\cdot V\cdot\overline{Z}+\overline{N}\cdot\overline{V}\cdot\overline{Z}$	符号ビット考慮される
LE	less or equal	1111	$Z+N\cdot\overline{V}+\overline{N}\cdot V$	符号ビット考慮される

付録 3. オペレーションコード表

オペレーション・コード・マップ

Bite 15 thru 12	オ ペ レ ー シ ョ ン
0000	Bit Manipolation/MOVEP/Immediate
0001	Move Byte
0010	Move Loog
0011	Move Word
0100	Miscellaneous
0101	ADDO/SUBO/Scc/DBcc
0110	Bcc,BSR
0111	MOVEQ
1000	OR/DIV/SBCD
1001	SUB/SUBX
1010	(Unassigned)
1011	CMP/EOR
1100	AND/MUL/ABCD/EXG
1101	ADD/ADDX
1110	Shift/Rotate
1111	(Unassigned)

Dynamic Bit

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	Register			1	Type		Effective Address					

Static Bit

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	Type		Effective Address					

Bit Type Codes:TST-00,CHG-01,CLR-10,SET-11

MOVEP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	Register			Op Mode			0	0	1	Register		

Op Mode:Word to Reg-100,Long to Reg-101,
Word to Mem-110,Long to Mem-111

OR Immediate

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	Size		Effective Address					

AND Immediate

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	Size		Effective Address					

SUB Immediate

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	0	Size		Effective Address					

ADD Immediate

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	1	0	Size		Effective Address					

EOR Immediate

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	Size		Effectiye Address					

CMP Immediate

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	0	Size		Effectiye Address					

MOVE Byte

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	Destination Register			Destination Mode			Sourco Mode			Sourco Register		

MOVE Long

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	Destination Register			Destination Mode			Source Mode			Source Register		

MOVE Word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	Destination Register			Destination Mode			Source Mode			Source Register		

NEGX

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	Size		Effectiye Address					

MOVE from SR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	1	1	Effectiye Address					

CLR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	1	0	Size		Effective Address					

NEG

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	0	0	Size		Effective Address					

MOVE to CCR

15	114	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	0	0	1	1	Effective Address					

NOT

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	1	0	Size		Effective Address					

MOVE to SR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	1	0	1	1	Effective Address					

NBCD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	0	Effective Address					

PEA

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	1	Effective Address					

SWAP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	1	0	0	0	Register		

MOVEM Registers to EA

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	1	Sz	Effective Address					

Sz:Long-1,WOrd-0

EXTW

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	1	0	0	0	0	Register		

EXTL

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	1	1	0	0	0	Register		

TST

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0	Size		Effective Address					

TAS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0	1	1	Effective Address					

MOVEM EA to Registers

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	1	Sz	Efffctiye Address					

Sz:Long-1,Word-0

TRAP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	0	Vector			

LINK

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	1	0	Register		

UNLK

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	1	1	Register		

MOVE to USP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	0	0	Register		

MOVE from USP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	0	1	Register		

RESET

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	0	0

NOP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	0	1

STOP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	1	0

RTE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	1	1

RTS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	0	1

TRAPV

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	1	0

RTR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	1	1

JSR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	1	0	Effectiye Address					

JMP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	1	1	Effectiye Address					

CHK

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	Register			1	1	0	Effectiye Address					

LEA

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	Register			1	1	1	Effectiye Address					

ADDQ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	Data			0	Size		Effectiye Address					

SUBQ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	Data			1	Sive		Effective Address					

Scc

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	Condition				1	1	Effective Address					

DBcc

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	Condition				1	1	0	0	1	Register		

Bcc

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	Condition				8 bit Displacement							

BSR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	1	8 bit Displacement							

MOVEO

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	Register			0	Data							

OR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	Register			Op Mode			Effectiye Address					

Op Mode

B	W	L	
000	001	010	Dn+EA→Dn
100	101	110	EA+Dn→EA

DIVU

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	Register			0	1	1	Effective Address					

DIVS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	Register			1	1	1	Effective Address					

SBCD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	Destination Register			1	0	0	0	0	R/M	Source Register		

R/M (register/memory):register·register-0,
memory·memory-1

SUB

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	Register			Op Mode			Effective Address					

Op Mode

.B	.W	.L	
000	001	010	Dn-EA→Dn
100	101	110	EA-Dn→EA
	011	111	An-EA→An

.SUBX

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	Destination Register			1	Size		0	0	R/M	Source Register		

R/M (register/memory):register·register-0,
memory·memory-1

CMP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	Register			Op Mode			Effective Address					

Op Mode

.B	.W	.L	
000	001	010	Dn EA
	011	111	An EA

CMPM

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	Register			1	Size		0	0	1	Register		

EOR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	Register			1	Size		Effective Address					

AND

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	
1	1	0	0	Register			Op mode			Effective Address					

Op Mode

.B	.W	.L	
000	001	010	Dn·EA→Dn
100	101	110	EA·Dn→EA

MULU

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	Register			0	1	1		Effective Address				

MULS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	Register			1	1	1		Effective Address				

ABCD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	Destination Register			1	0	0	0	0	R/M	Source Register		

R/M (register/memory):register·register-0,
memory·memory-1

EXGD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	Data Register			1	0	1	0	0	0	Data Register		

EXGA

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	Address Register			1	0	1	0	0	1	Address Register		

EXGM

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	Data Register			1	1	0	0	0	1	Address Register		

ADD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	Register			Op Mode			Effective Address					

Op Mode

.B	.W	.L	
000	001	010	Dn+EA→Dn
100	101	110	EA+Dn→EA
	011	111	An+EA→An

ADDX

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	Destination Register			1	Size		0	0	R/M	Source Register		

R/M (register/memory):register·register-0,
memory·memory-1

Data Register Shifts

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	Count/ Register			d	Size		I/r	Type		Register		

Memory Shifts

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	Type		d	1	1	Effective Address					

Shift Type Codes: AS-00, LS-01, ROX-10, RO-11
d (direction): Right-0, Left-1
t/r (count source): Immediate Count-0,
Register Count-1

付録 4.　電気的仕様とタイミング図——MC 68000 L 4・L 6・L 8・L 12

4・1　電気特性

絶対最大定格

定　　　格	記号	定格値	単位
供給電圧	V_{CC}	−0.3〜+7.0	V
入力電圧	V_{in}	−0.3〜+7.0	V
動作温度	T_A	0〜70	℃
保存温度	T_{stg}	−55〜+150	℃

温度特性

特　　　性	記号	特性値	単位
熱抵抗　セラミック・パッケージ	θ_{JA}	30	℃/W

このデバイスは，各入力に対する静電気あるいは高電圧に対する保護回路を備えていますが，応用上あらかじめハイ・インピーダンスの入力回路に，絶対最大定格を越えるような電圧がかからないように注意する必要があります．

未使用の入力ピンは，常に V_{SS} または V_{CC} に接続しなければなりません．

電　力　条　件

チップの接合点の平均温度 T_J(℃) は，次式で計算できます．

$$T_J = T_A + (P_D \cdot \theta_{JA}) \quad (1)$$

ここで

T_A≡周囲温度（℃）

θ_{JA}≡パッケージの熱抵抗，ジャンクション-周囲間（℃/W）

$P_D \equiv P_{INT} + P_{I/O}$

$P_{INT} \equiv I_{CC} \times V_{CC}$ (W)——チップの内部消費電力

$P_{I/O}$ 入/出力端子でのI/O 消費電力（W）——ユーザの使用方法による．

一般の使用条件では，$P_{I/O} \ll P_{INT}$ のため無視できます．

P_{PORT} が無視できる場合の P_D と T_J の近代的な関係は，

$$P_D = K \div (T_J + 273\,^\circ\mathrm{C}) \qquad (2)$$

となります．(1)，(2)式からKを求めると，

$$K = P_D \cdot (T_A + 273\,^\circ\mathrm{C}) + \theta_{JA} \cdot P_D^2 \qquad (3)$$

ここでKは既知のT_Aにおける平衡状態のP_Dを測定すれば，(3)式によって決定できる定数です．このKの値を(1)，(2)式に代入し，演算をすることによって，任意のT_AにおけるP^DとT_Jを計算することができます．

DC電気的特性（V_{CC}=5.0 Vdc±5%，V_{SS}=0 Vdc；T_A=0°C～70°C）

特性		記号	Min	Max	単位
入力“H”電圧		V_{IH}	2.0	V_{CC}	V
入力“L”電圧		V_{IL}	$V_{SS}-0.3$	0.8	V
入力リーク電流 @5.25 V	$\overline{BERR}$, $\overline{BGACK}$, $\overline{BR}$, $\overline{DTACK}$, CLK, $\overline{IPL0}$-$\overline{IPL2}$, $\overline{VPA}$ $\overline{HALT}$, $\overline{RESET}$	I_{in}	— —	2.5 20	μA
スリー・ステート（オフ・ステート）入力電流 @2.4 V/0.4 V	$\overline{AS}$, A1-A23, D0-D15 FC0-FC2, $\overline{LDS}$, R/$\overline{W}$, $\overline{UDS}$, $\overline{VMA}$	I_{TSI}	—	20	μA
出力“H”電圧（I_{OH}=−400 μA）	E* $\overline{AS}$, A1-A23, BG, D0-D15 FC0-FC2, $\overline{LDS}$, R/$\overline{W}$, $\overline{UDS}$, $\overline{VMA}$	V_{OH}	$V_{CC}-0.75$ 2.4	— —	V
出力“L”電圧 （I_{OL}=1.6 mA） （I_{OL}=3.2 mA） （I_{OL}=35.0 mA） （I_{OL}=5.3 mA）	$\overline{HALT}$ A1-A23, $\overline{BG}$, FC0-FC2 $\overline{RESET}$ E, $\overline{AS}$, D0-D15, $\overline{LDS}$, R/$\overline{W}$ $\overline{UDS}$, $\overline{VMA}$	V_{OL}	— — — —	0.5 0.5 0.5 0.5	V
消費電力（クロック周波数=8 MHz）		P_D	—	1.5	W
入力容量（V_{in}=0 V，T_A=25°C；周波数=1 MHz）		C_{in}	—	10.0	pF

＊　470 Ω のプルアップ抵抗を外付

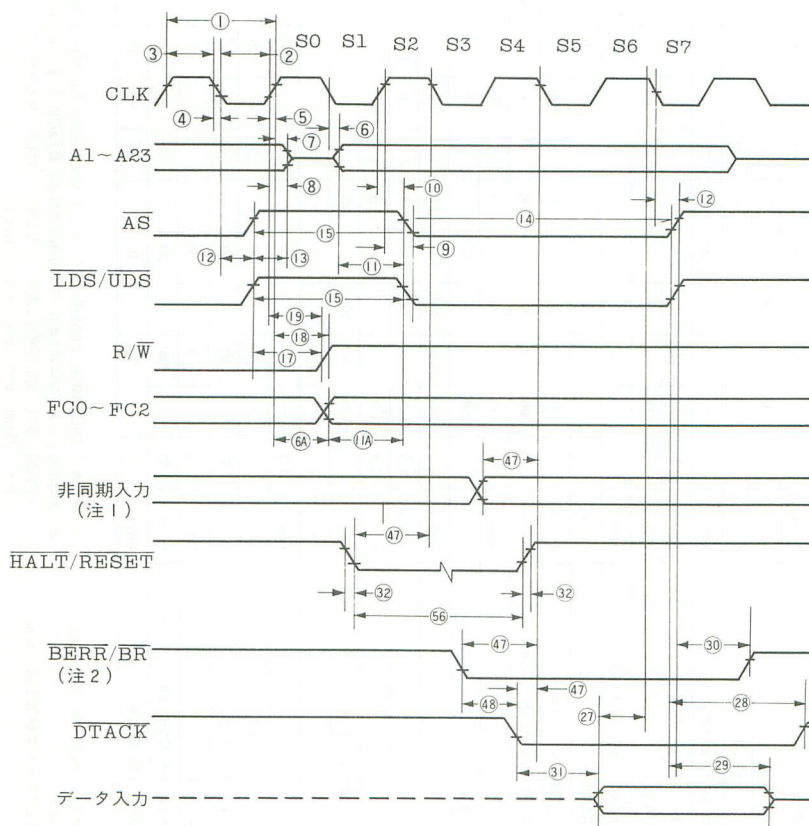
4・2 AC電気的特性（V_{CC}=5.0 Vdc±5 %, V_{SS}=0 Vdc ; T_A=0 ℃～70 ℃）

番号	特性	記号	4 MHz MC 68000 L 4		6 MHz MC 68000 L 6		8 MHz MC 68000 L 8		10 MHz MC 68000 L 10		12.5 MHz MC 68000 L 12		単位
			Min	Max	Min	Max	Min	Max	Min	Max	Max	Max	
1	Clock Period	t_{CYC}	250	500	167	500	125	500	100	500	80	250	ns
2	Clock Width Low	t_{CL}	115	250	75	250	55	250	45	250	35	125	ns
3	Clock Width High	t_{CH}	115	250	75	250	55	250	45	250	35	125	ns
4	Clock Fall Time	t_{Cf}	—	10	—	10	—	10	—	10	—	5	ns
5	Clock Rise Time	t_{Cr}	—	10	—	10	—	10	—	10	—	5	ns
6	Clock Low to Address	t_{CLAV}	—	90	—	80	—	70	—	55	—	55	ns
6A	Clock High to FC Valid	t_{CHFCV}	—	90	—	80	—	70	—	60	—	55	ns
7	Clock to Address Data High Impedance (Maximum)	t_{CHAZx}	—	120	—	100	—	80	—	70	—	60	ns
8	Clock High to Address/FC Invalid (Minimum)	t_{CHAZn}	0	—	0	—	0	—	0	—	0	—	ns
9[1]	Clock High to $\overline{AS}$,$\overline{DS}$Low (Maximum)	t_{CHSLx}	—	80	—	70	—	60	—	55	—	55	ns
10	Clock High to $\overline{AS}$,$\overline{DS}$ Low (Minimum)	t_{CHSLn}	0	—	0	—	0	—	0	—	0	—	ns
11[2]	Address to $\overline{AS}$,$\overline{DS}$ (Read)Low/$\overline{AS}$ Write	t_{AVSL}	55	—	35	—	30	—	20	—	0	—	ns
11A[2]	FC Valid to $\overline{AS}$,$\overline{DS}$ (Read)Low/$\overline{AS}$ Write	t_{FCVSL}	80	—	70	—	60	—	50	—	40	—	ns
12[1]	Clock Low to $\overline{AS}$,$\overline{DS}$ High	t_{CLSH}	—	90	—	80	—	70	—	55	—	50	ns
13[2]	$\overline{AS}$,$\overline{DS}$ High to Address/FC Invalid	t_{SHAZ}	60	—	40	—	30	—	20	—	10	—	ns
14[2,5]	$\overline{AS}$,$\overline{DS}$ Width Low(Read)/$\overline{AS}$ Write	t_{SL}	535	—	337	—	240	—	195	—	160	—	ns
14A[2]	DS Width Low (Write)	—	285	—	170	—	115	—	95	—	80	—	ns
15[2]	$\overline{AS}$,$\overline{DS}$ Width High	t_{SH}	285	—	180	—	150	—	105	—	65	—	ns
16	Clock High to AS,DS High Impedance	t_{CHSZ}	—	120	—	100	—	80	—	70	—	60	ns
17[2]	$\overline{AS}$,$\overline{DS}$ High to R/$\overline{W}$ High	t_{SHRH}	60	—	50	—	40	—	20	—	10	—	ns
18[1]	Clock High to R/$\overline{W}$ High (Maximum)	t_{CHRHx}	—	90	—	80	—	70	—	60	—	60	ns
19	Clock High to R/$\overline{W}$ High (Minimum)	t_{CHRHn}	0	—	0	—	0	—	0	—	0	—	ns
20[1]	Clock High to R/$\overline{W}$ Low	t_{CHRL}	—	90	—	80	—	70	—	60	—	60	ns
21[2]	Address Valid to R/$\overline{W}$ Low	t_{AVRL}	45	—	25	—	20	—	0	—	0	—	ns

21A[2]	FC Valid to R/$\overline{W}$ Low	t_{FCVRL}	80	—	70	—	60	—	50	—	30	—	ns
22[2]	R/$\overline{W}$ Low to $\overline{DS}$ Low (Write)	t_{RLSL}	200	—	140	—	80	—	50	—	30	—	ns
23	Clock Low to Data Out Valid	t_{CLDO}	—	90	—	80	—	70	—	55	—	55	ns
25[2]	$\overline{DS}$ High to Data Out Invalid	t_{SHDO}	60	—	40	—	30	—	20	—	15	—	ns
26[2]	Data Out Valid to $\overline{DS}$ Low (Write)	t_{DOSL}	55	—	35	—	30	—	20	—	15	—	ns
27[6]	Data In to Clock Low (Setup Time)	t_{DICL}	30	—	25	—	15	—	15	—	15	—	ns
28[2]	$\overline{AS}$,$\overline{DS}$ High to $\overline{DTACK}$ High	t_{SHDAH}	0	240	0	160	0	120	0	90	0	70	ns
29	$\overline{DS}$ High to Data Invalid (Hold Time)	t_{SHDI}	0	—	0	—	0	—	0	—	0	—	ns
30	$\overline{AS}$,$\overline{DS}$ High to $\overline{BERR}$ High	t_{SHBEH}	0	—	0	—	0	—	0	—	0	—	ns
31[2,6]	$\overline{DTACK}$ Low to Data In (Setup Time)	t_{DALDI}	—	180	—	120	—	90	—	65	—	50	ns
32	$\overline{HALT}$ and $\overline{RESET}$ Input Transition Time	t_{RHrf}	0	200	0	200	0	200	0	200	0	200	ns
33	Clock High to $\overline{BG}$ Low	t_{CHGL}	—	90	—	80	—	70	—	60	—	50	ns
34	Clock High to $\overline{BG}$ High	t_{CHGH}	—	90	—	80	—	70	—	60	—	50	ns
35	$\overline{BR}$ Low to $\overline{BG}$ Low	t_{BRLGL}	1.5	3.0	1.5	3.0	1.5	3.0	1.5	3.0	1.5	3.0	Clk. Per.
36	$\overline{BR}$ High to $\overline{BG}$ High	t_{BRHGH}	1.5	3.0	1.5	3.0	1.5	3.0	1.5	3.0	1.5	3.0	Clk. Per.
37	$\overline{BTACK}$ Low to $\overline{BG}$ High	t_{GALGH}	1.5	3.0	1.5	3.0	1.5	3.0	1.5	3.0	1.5	3.0	Clk. Per.
38	$\overline{BG}$ Low to Bus High Impedance(With $\overline{AS}$ High)	t_{GLZ}	—	120	—	100	—	80	—	70	—	60	ns
39	$\overline{BG}$ Width High	t_{GH}	1.5	—	1.5	—	1.5	—	1.5	—	1.5	—	Clk. Per.
46	$\overline{BGACK}$ Width	t_{BGL}	1.5	—	1.5	—	1.5	—	1.5	—	1.5	—	Clk. Per.
47[6]	Asynchronous Input Setup Time	t_{ASI}	30	—	25	—	20	—	20	—	20	—	ns
48	$\overline{BERR}$ Low to $\overline{DTACK}$ Low (Note 3)	t_{BELDAL}	50	—	50	—	50	—	50	—	50	—	ns
53	Data Hold from Clock High	t_{CHDO}	0	—	0	—	0	—	0	—	0	—	ns
55	R/$\overline{W}$ to Data Bus Impedance Change	t_{RLDO}	55	—	35	—	30	—	20	—	10	—	ns
56	Halt/$\overline{RESET}$ Pulse Width (Note 4)	t_{HRPW}	10	—	10	—	10	—	10	—	10	—	

注：

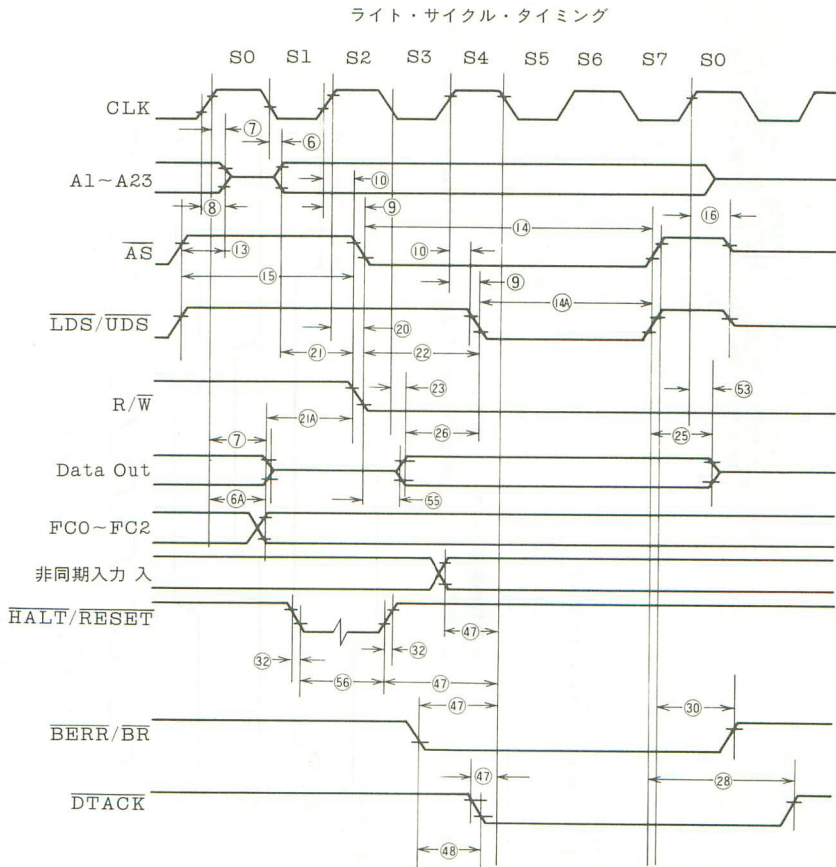
1．50 pF 以下の負荷の場合は，表の値から 5 ns 差し引いた値となる．
2．実際の値はクロック周期により異なる．
3．♯47 が $\overline{DTACK}$，$\overline{BERR}$ で共に満足されると ♯48 は 0 ns となる．
4．V_{CC} が供給されて 100 ms 後．
5．♯14A は T6E，BF4，R9M のマスク・セットでは表の値より 1 クロック周期少ない．
6．非同期セットアップ時間（♯47）要求が満たされると $\overline{DTACK}$ が“L”からデータのセットアップ時間（♯31）要求は無視できる．そのときデータはデータ入力からクロック“L”のセットアップ時間（♯27）が満たされていればよい．

4・3 リード・サイクルのタイミング

注：

1．非同期入力 $\overline{BGACK}$，$\overline{IPL0}$～$\overline{IPL2}$，$\overline{VPA}$ は次のクロックの立ち下がりで検出されます．そのためにはセット・アップ時間が必要である．

2．$\overline{BR}$ はこのバス・サイクルの終りで認識される必要があるときのみ，このタイミングでアサートされる必要がある．

3．特記なき場合，すべての入力，出力波形は“H”電圧＝2.0 V，“L”電圧＝0.8 V を測定点として規定される．

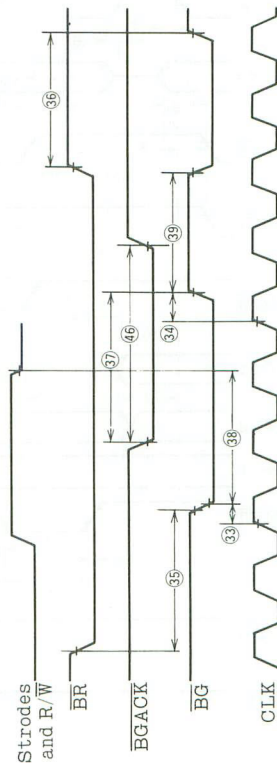
4・4　ライト・サイクルのタイミング

注：特記なき場合，すべての入力，出力波形は"H"電圧=2.0 V，"L電圧=0.8 V を測定点として規定される．

4・5 バス・アービトレーション

AC 電気的特性――バス・アービトレーション（V_{CC}=5.0 Vdc±5％, V_{SS}=0 Vdc；T_A=0°C～70°C, 下図参照）

番号	特性	記号	4 MHz MC 68000 L 4		6 MHz MC 68000 L 6		8 MHz MC 68000 L 8		10 MHz MC 68000 L 10		12.5 MHz MC 68000 L 12		単位
			Min	Max	Min	Max	Min	Max	Min	Max	Min	Max	
33	Clock High to $\overline{BG}$ Low	t_{CHGL}	—	90	—	80	—	70	—	60	—	50	ns
34	Clock High to $\overline{BG}$ High	t_{CHGH}	—	90	—	80	—	70	—	60	—	50	ns
35	$\overline{BR}$ Low to $\overline{BG}$ Low	t_{BRLGL}	1.5	3.5	1.5	3.5	1.5	3.5	1.5	3.5	1.5	3.0	Clk. Per.
36	$\overline{BR}$ High to $\overline{BG}$ High	t_{BRHGH}	1.5	3.0	1.5	3.0	1.5	3.0	1.5	3.0	1.5	3.0	Clk. Per.
37	$\overline{BGACK}$ Low to $\overline{BG}$ High	t_{GALGH}	1.5	3.0	1.5	3.0	1.5	3.0	1.5	3.0	1.5	3.0	Clk. Per.
38	$\overline{BG}$ Low to Bus High Impedance (With $\overline{AS}$ High)	t_{GLZ}	—	120	—	100	—	80	—	70	—	60	ns
39	$\overline{BG}$ Width High	t_{GH}	1.5	—	1.5	—	1.5	—	1.5	—	1.5	—	Clk. Per.
46	$\overline{BGACK}$ Width	t_{BGL}	1.5	—	1.5	—	1.5	—	1.5	—	1.5	—	Clk. Per.

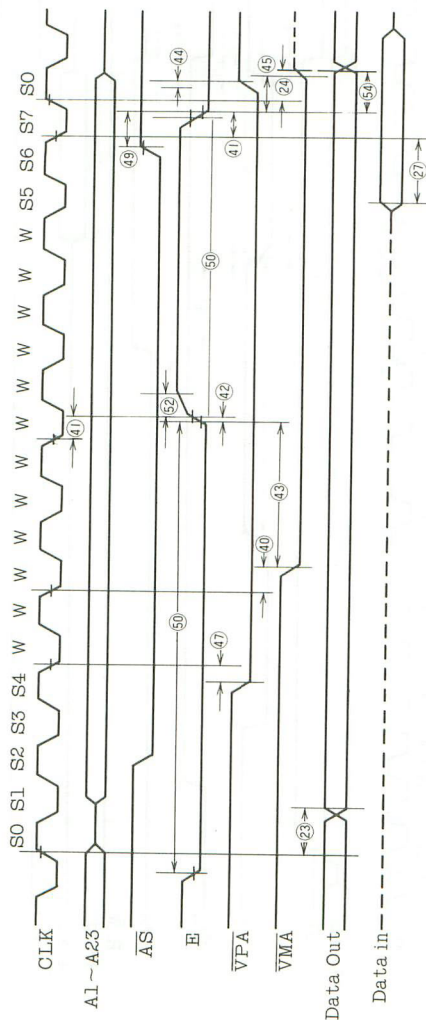
―― AC 電気的波形―バス・アービトレーション ――

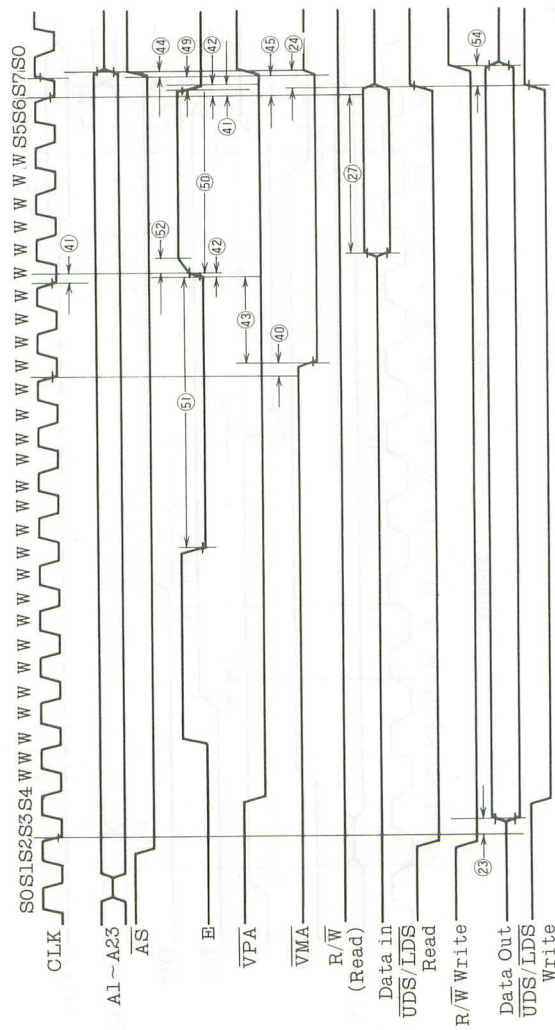
この波形はタイミング規定のエッジからエッジの測定用のものであり入出力信号の機能を説明するものではない．他の機能説明とそれに関するデバイス動作の図も参照されたい．

注：

1．非同期入力$\overline{BERR}$, $\overline{BGACK}$, $\overline{BR}$, DTACK, $\overline{IPL0}$～$\overline{IPL2}$．$\overline{VPA}$は次のクロックの立ち下がりで検出される．そのためには，セットアップ時間が必要である．

2．すべての入力，出力の波形は“H”電圧=2.0 V,“L”電圧=0.8 V を測定点として規定される．

4・6　最良タイミング・ケース

M6800タイミング-ベスト・ケース

注：この図はM 6800の最良のタイミングを示し，Eの立ち下がり後3つ目のシステム・クロック・サイクルの前にVPAが下がっている．

4·7 最悪タイミング・ケース

参考文献

（モトローラ社提供）

1）MC 68000 データブック
2）MC 68000 マイクロプロセッサ・ユーザーズ・マニアル
3）M 68000 リアルタイム・マルチタスキング・ソフトウェア・ユーザーズ・ガイド
4）M 68000 EXORmacs 開発システム操作マニアル
5）スピード・マスタ 68 K 取扱い説明書
6）M 68000 ユーザ・システム・エミュレータ・ユーザーズ・ガイド
7）M 68000 常駐ストラクチャード・アセンブラ・リファレンス・マニアル
8）VERSAdos I/O およびファイル管理サービス・マニアル
9）M 68000/IPC コマンドチャンネル・ソフトウェア・インターフェス・リファレンス・マニアル
10）M 68000 リンケージ・エディタ・ユーザーズ・マニアル
11）マルチ・チャンネル・コミニュケーション・モジュール・ユーザス・ガイド
12）M 68 KBSA リアルタイム・バス・アナライザー・ユーザス・ガイド

索　　引

［ア　行］

［カ　行］

［サ　行］

［タ　行］

［ナ　行］

［ハ　行］

索　　　　　引

［マ　行］

［ヤ　行］

［ラ　行］

［ワ　行］

著者略歴

昭和42年　法政大学工学部電気工学科卒
現　　在　C & IT(株)代表取締役社長
著　　書　「パソコン入門心得帖」(共著;オーム社)

図解16ビットマイクロコンピュータ
MC68000の使い方

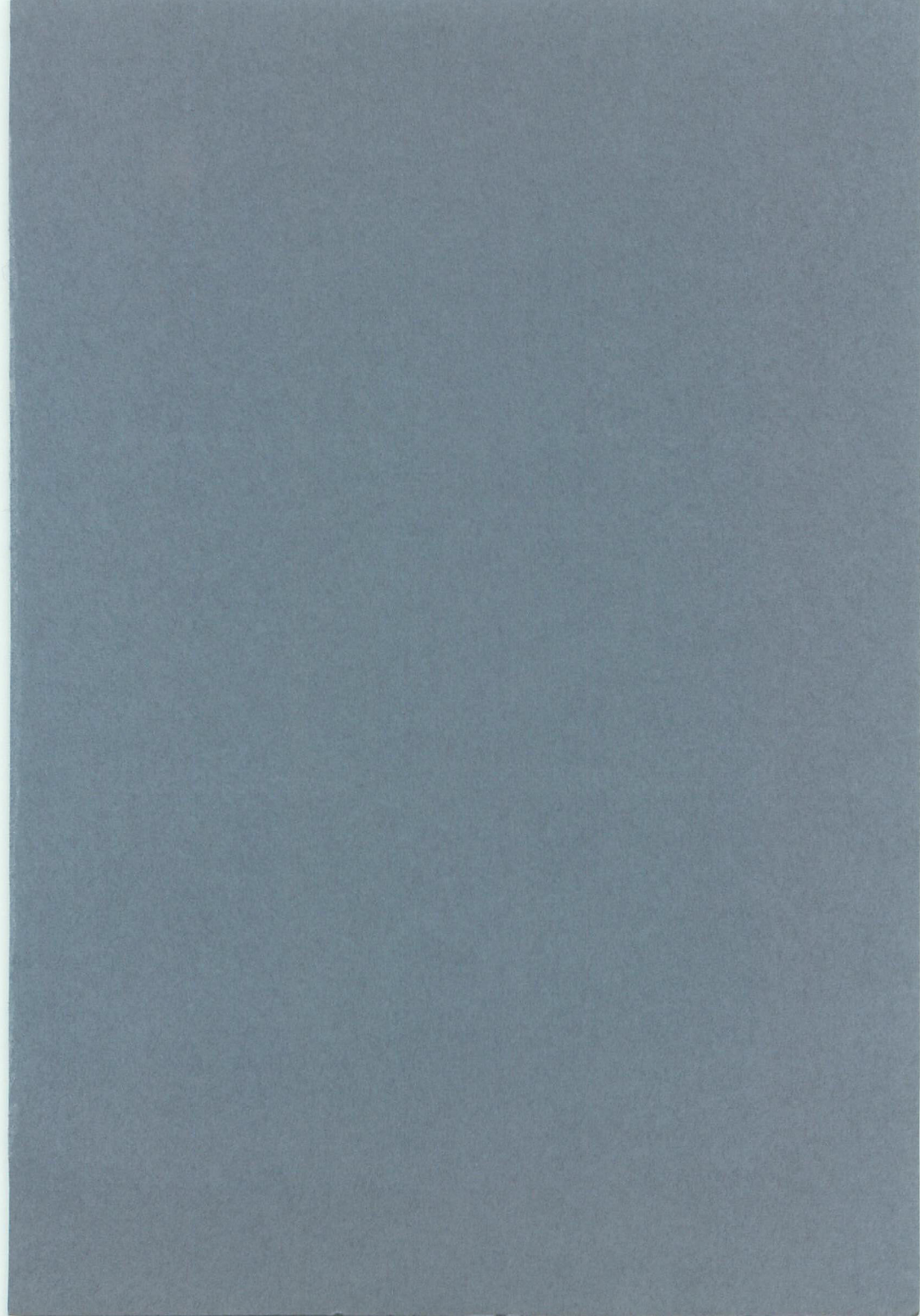
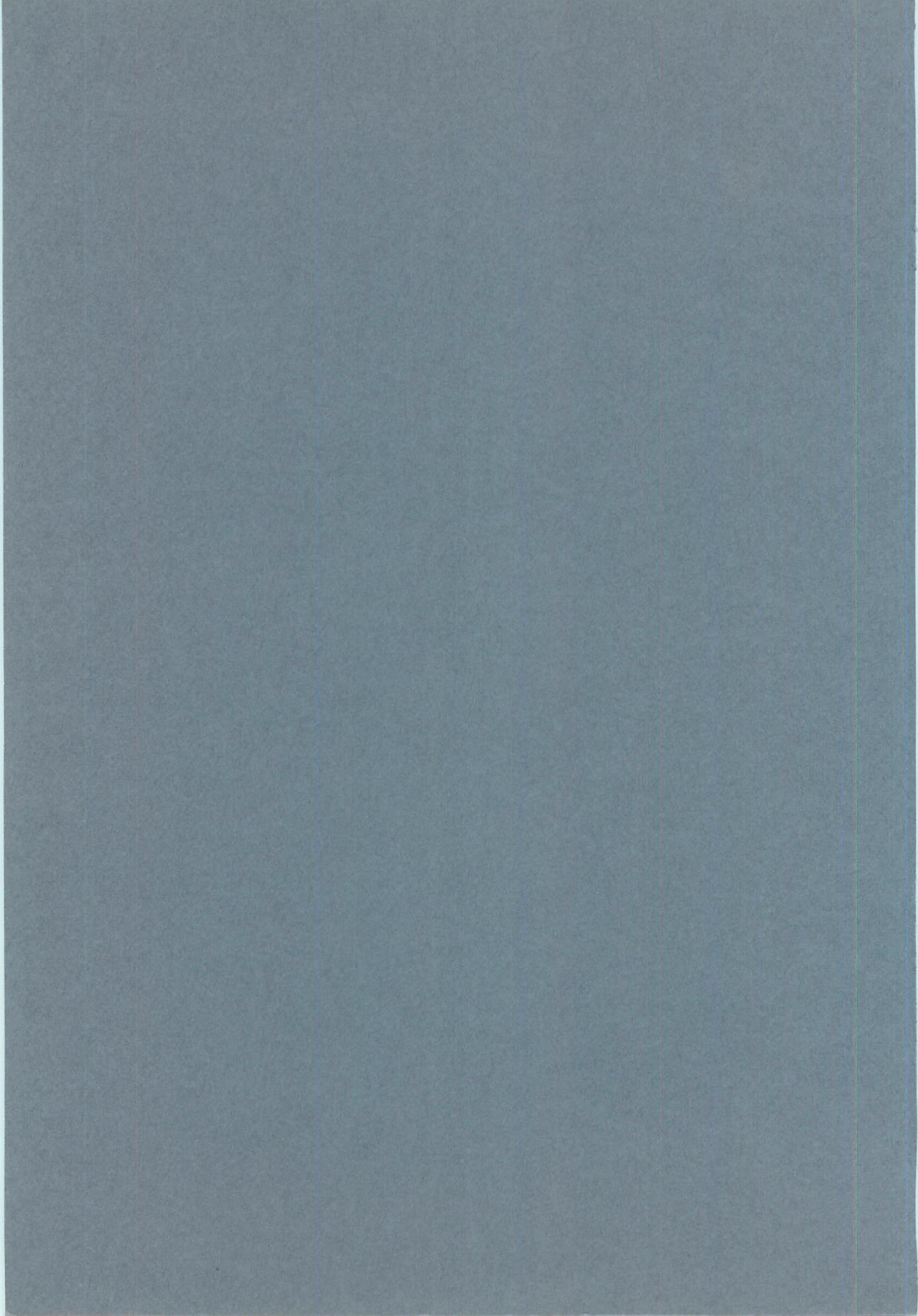
昭和58年10月1日　第1版第1刷発行
平成4年5月20日　第1版第7刷発行

OHM·OHM·OHM·
著者承認
検印省略

著　者　小島　進
発行者　株式会社　オーム社
代表者　佐藤政次
発行所　株式会社　オーム社
郵便番号　101
東京都千代田区神田錦町3-1
振替　東京6-20018
電話　03(3233)0641(代表)

Printed in Japan

印刷　中央印刷　製本　関川製本所
落丁・乱丁本はお取替えいたします

ISBN 4-274-07162-6

図解コンピュータシリーズ
江村潤朗 監修

コンピュータシステム入門
（改訂２版）江村潤朗・野津　昭 共著（A5・p.268）
ハードウェア・システム入門
江村潤朗 著（A5・p.216）
FORTRAN 入門
海老沢成享 著（A5・p.230）
COBOL 入門
海老沢成享 著（A5・p.350）
PL／I 入門
光吉民恵 著（A5・p.244）
PASCAL プログラミング入門
三沢常男・市川隆男 共著（A5・p.264）
BASIC プログラミング入門
平山静夫 著（A5・p.290）
アセンブラプログラミング入門
瀬下孝之・前田忠彦 共著（B5・p.380）
APL 入門
金子章弘 著（A5・p.236）
日本語 APL 入門
平尾隆行 著（A5・p.236）
簡易照会言語入門
―関係データベースシステム―
平尾隆行 著（A5・p.228）
ストラクチャード・プログラミング入門
（改訂２版）國友義久 著（A5・p. 258）
オペレーティング・システム入門
江村潤朗 著（A5・p.176）
ファイル編成入門
山谷正己 著（A5・p.184）
データベース入門
（改訂２版）穂鷹良介 著（A5・p.228）
仮想記憶システム入門
山谷正己 著（A5・p.140）
エキスパートシステム入門
平尾隆行 著（A5・p.176）
情報通信システム入門
八島朝一 著（A5・p.244）
データ通信システム入門
（改訂２版）保坂岩男 原著・石坂充弘 著（A5・p.270）
EDP システム設計入門
南條　優 著（A5・p.204）
オフィスコンピュータ入門
魚田勝臣・富沢研三 共著（A5・p.202）
オフィスオートメーション入門
（改訂２版）中村　茂 著（A5・p.238）
RPG プログラミング技法
野口正雄 著（B5・p.232）
プログラム流れ図の作成技法
江村潤朗・野津　昭 共著（B5・p.358）
対話式計算システムの活用技法
平尾隆行 著（B5・p.262）
プログラム開発管理
（改訂２版）國友義久 著（B5・p.218）
仮想計算機システム
山谷正己 著（B5・p.304）
多重仮想記憶オペレーティングシステム
鎌田　肇 著（B5・p.228）
コンピュータとアプリケーション
寺沢康夫 著（B5・p.236）
データベース／データ通信プログラミング
平尾隆行 著（B5・p.220）

カバー印刷：池田印刷